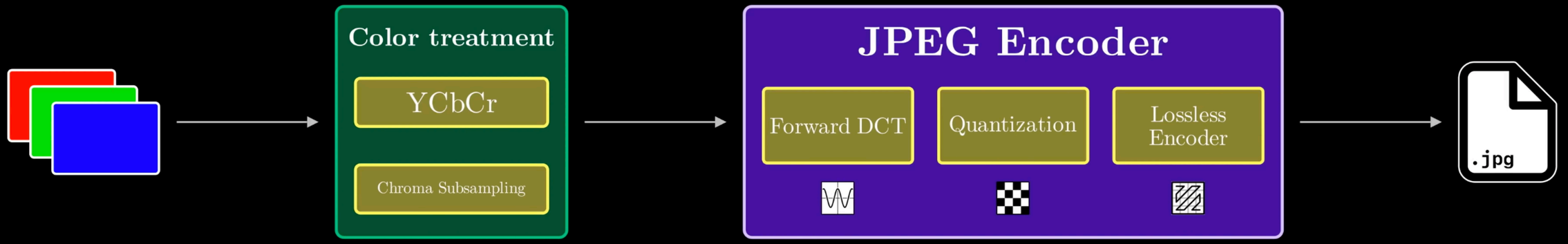
Learned Image Compression
EE274

Recap -> JPEG


Color treatment
YCbCr
Chroma Subsampling
JPEG Encoder
Forward DCT
Quantization
Lossless Encoder
.jpg

Recap -> BPG

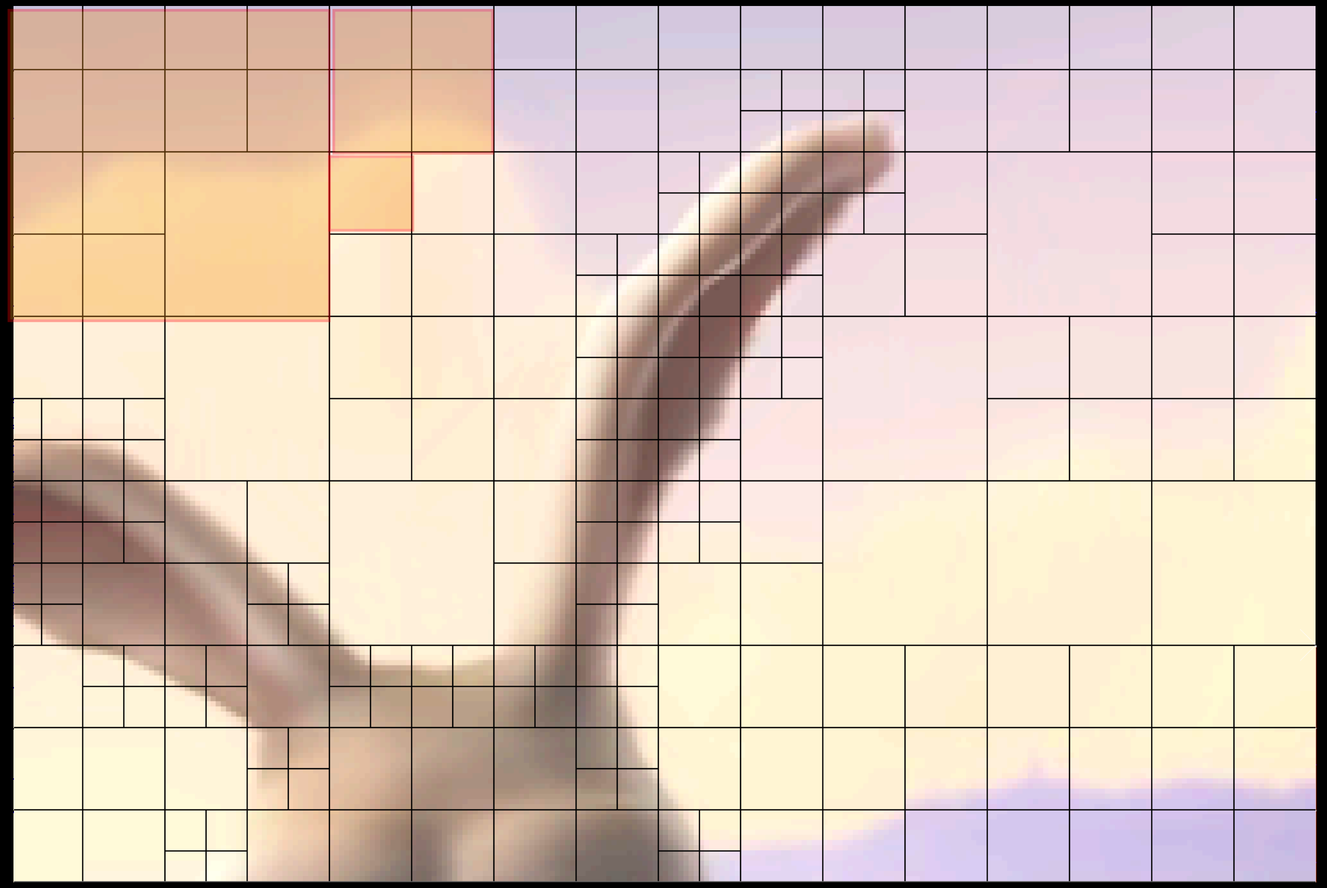


Image Compression -> JPEG 137x

Uncompressed -> 1.1MB
JPEG -> 8KB (~137x!)

Image from Kodak dataset

Image Compression -> BPG

Uncompressed -> 1.1MB
BPG -> 8KB (~137x!)

Image from Kodak dataset

What next?

- Beyond Linear transform: JPEG/JPEG2000/BPG all use variants of DCT, DWT etc. can we obtain better performance with non-linear transforms

- End-to-End RD Optimization: JPEG the R-D optimization is not accurate. Rate needs to be shared between different channels etc. Can we make that end-to-end?

https://wave-one.github.io/iframe_comparisons/

Recent Learned Image/Video Codec Works

- **Learned Image Compression:**
  [Toderici, CVPR15], [Theis, ICLR17], [Agustsson, NIPS17], [Baig, NIPS17],
  [Balle, ICLR17], [Rippel, ICML17], [Balle, ICLR18], [Johnston, CVPR18],
  [Mentzer, CVPR18], [Choi, ICLR19], [Balle, ICLR19], [Lee, ICLR19], [Mentzer, CVPR19]
  [Lu, 2021], [Ma et al, 2021], [Yang et al, NIPS2021], [Mentzer et al. NIPS2021] . . .

- **Learned Video Compression**
  [Wu. et al, ECCV18], [Lu et al, CVPR19], [Cheng et al, CVPR19]
  [Rippel et al ICCV19], [Hu et al, 2020], [Agustsson et al. 2020], [Golinski et al. 2020]
  [Habiban et al.2019], [Lu et al. 2020], [Liu el.al.2020], [DVC, Lu et al. 2019] . . .

- **The CLIC Challenge**
  “Challenge on Learned Image Compression”
  -> ongoing image compression contest at CVPR

Lots of interesting works!

Our Goal: Understand the Key concepts

The Image Compression Problem

I

Target Image

Encoder

bits

Decoder

$\hat{I}$

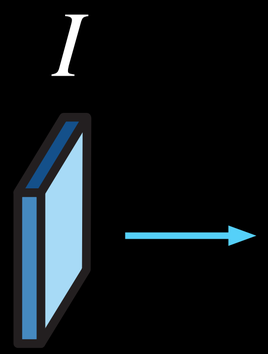
Target reconst

Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Rate

Distortion

Traditional Image Codecs

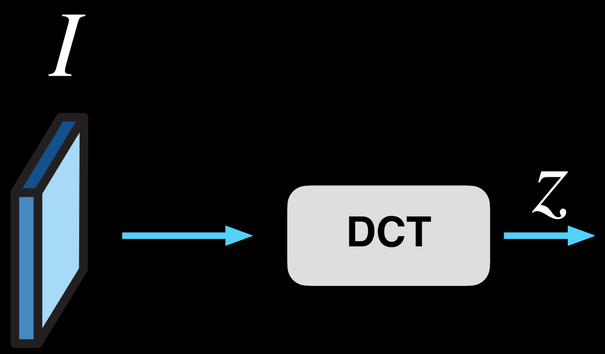


Target Image

Encoding proceeds in 3 steps:

CAUTION: Simplified framework

Traditional Image Codecs



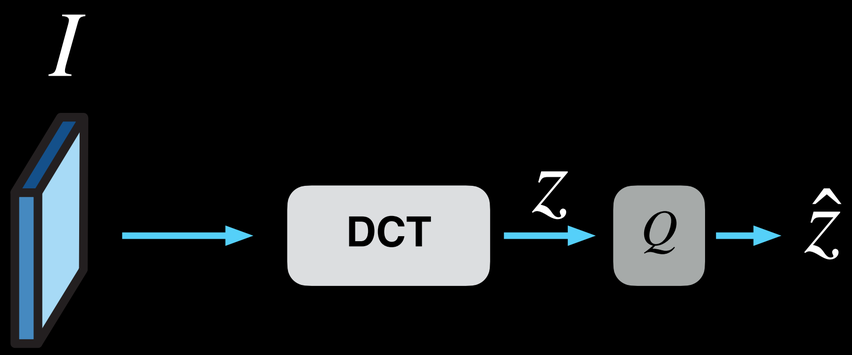
Target Image

Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels

CAUTION: Simplified framework

Traditional Image Codecs



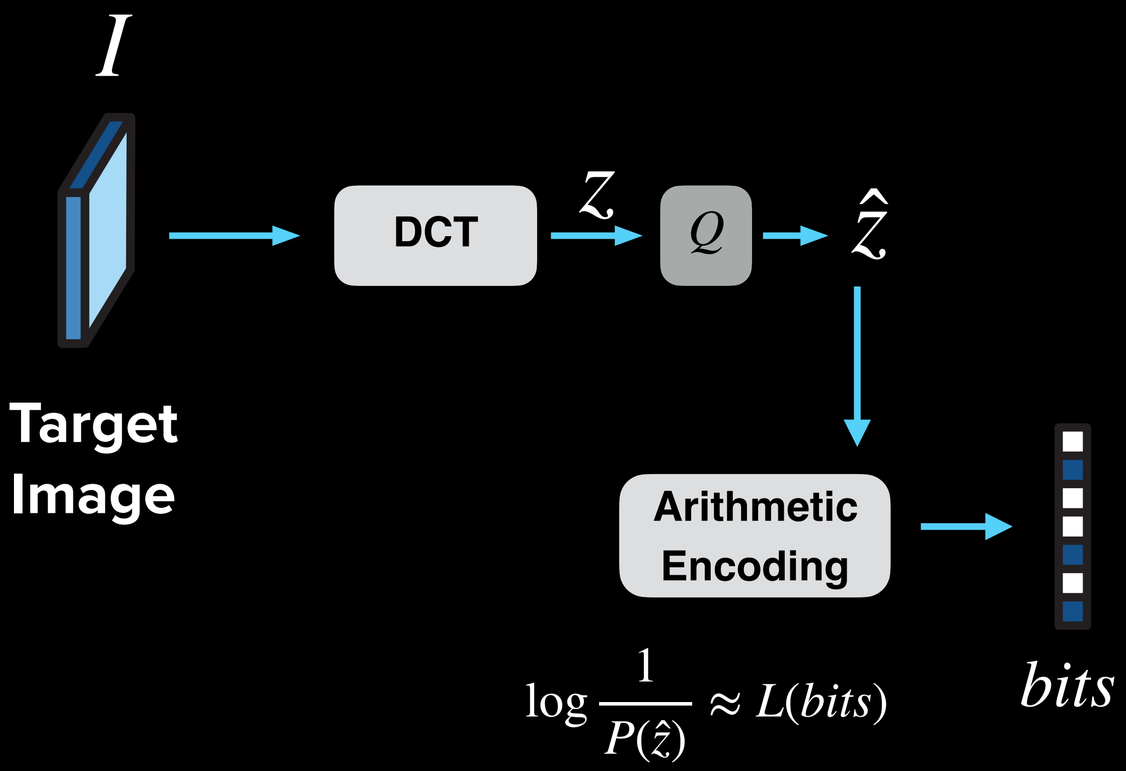
Target Image

Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels
- **Quantize ->** Loss of precision
  $Q([2.3,3.7]) = [2,4]$

CAUTION: Simplified framework

Traditional Image Codecs



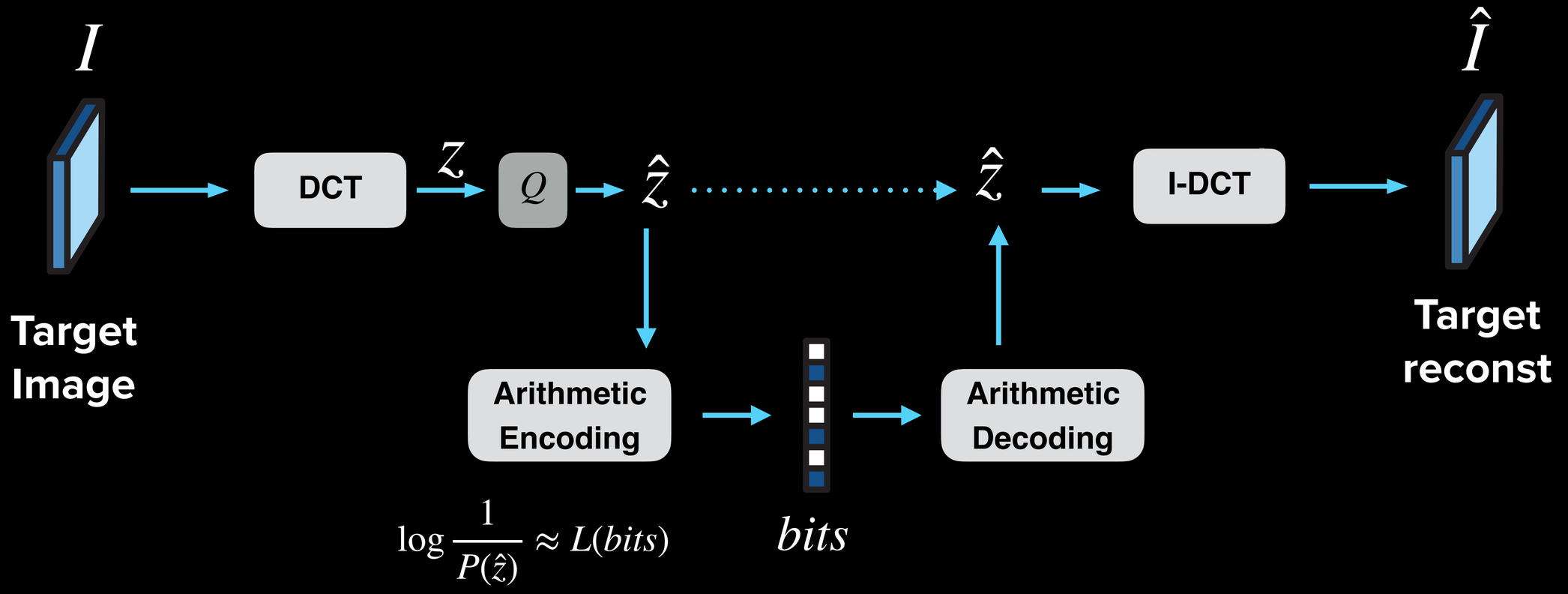
Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels
- **Quantize ->** Loss of precision
  $Q([2.3,3.7]) = [2,4]$
- **Arithmetic/Huffman ->** Lossless Compression
  In the simplest form, uses a discrete distribution $P(\hat{z})$ to encode $\hat{z}$ into a bitstream of length

$$L(bits) \approx \log \frac{1}{P(\hat{z})}$$

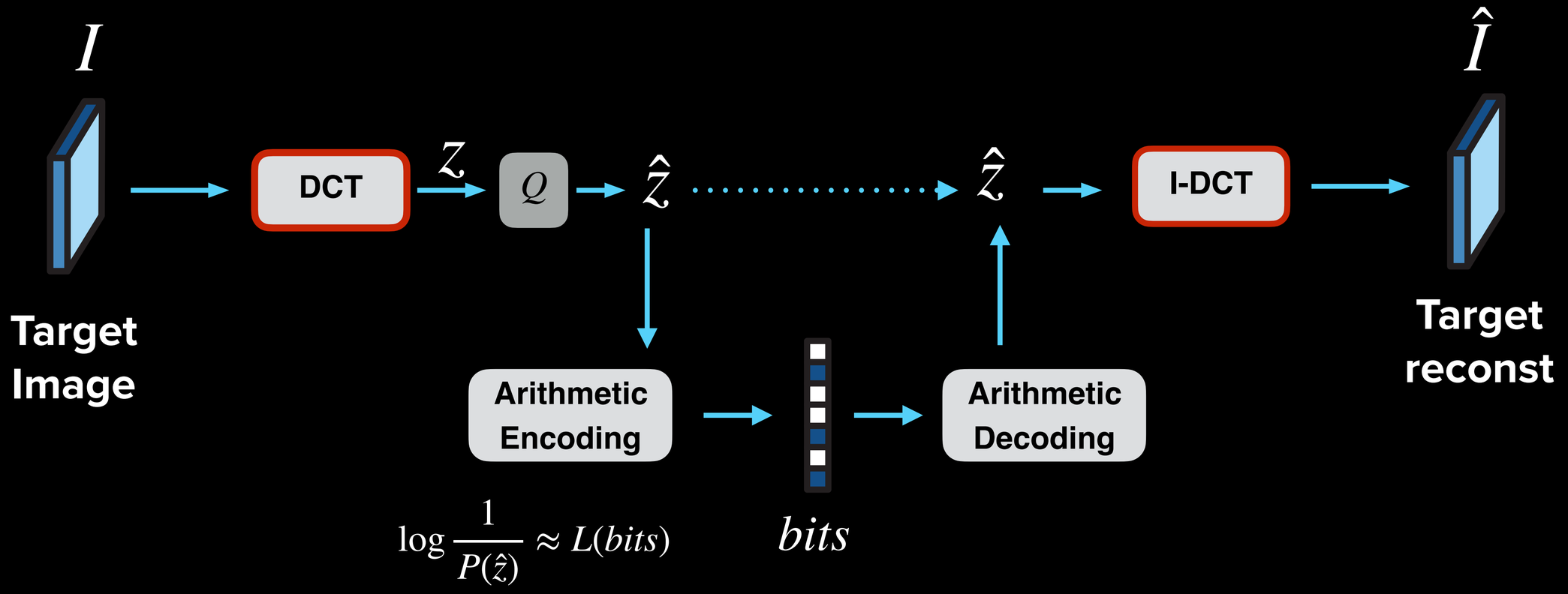
CAUTION: Simplified framework

Traditional Image Codecs



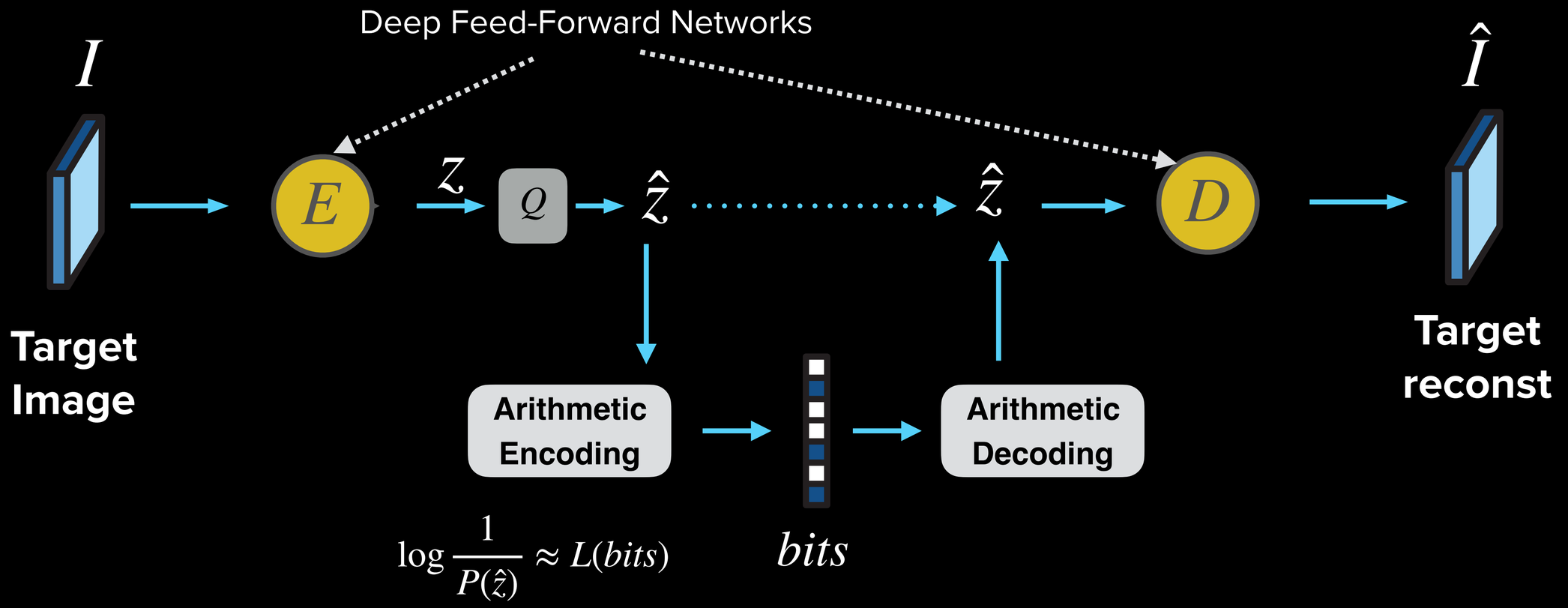
Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Traditional Image Codecs

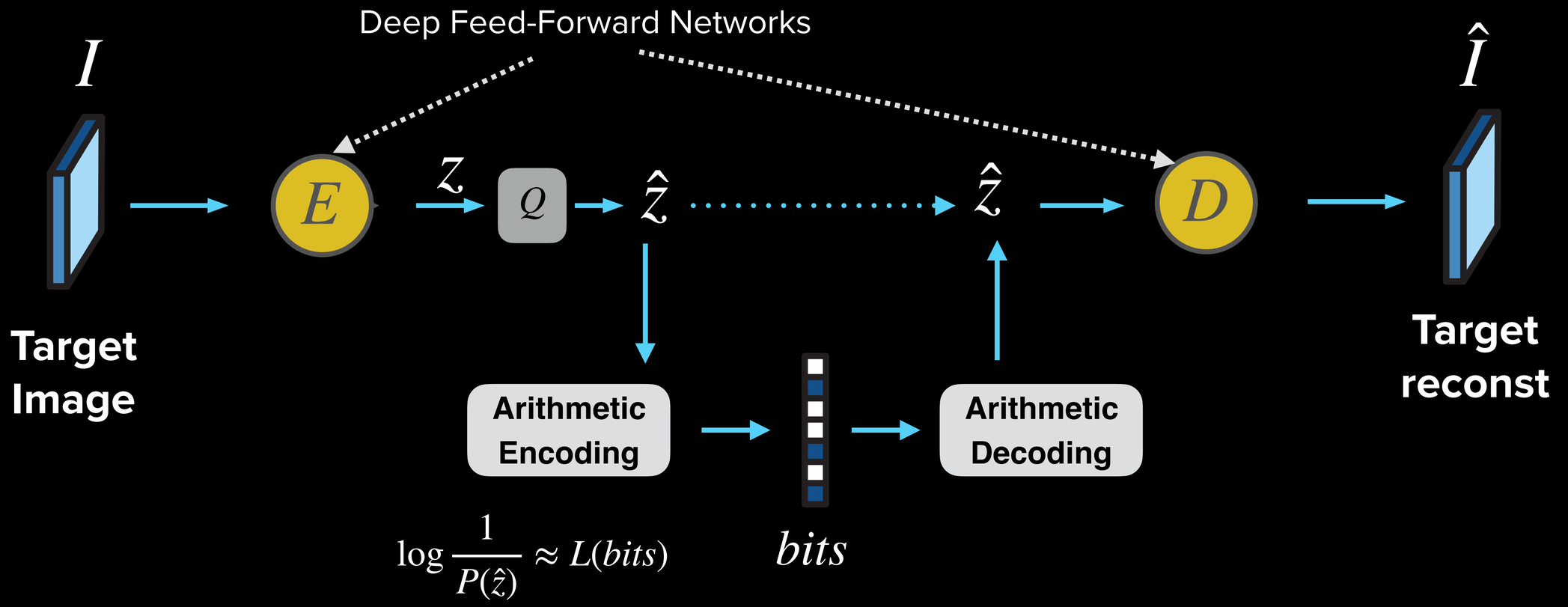


Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Learned Image Codecs

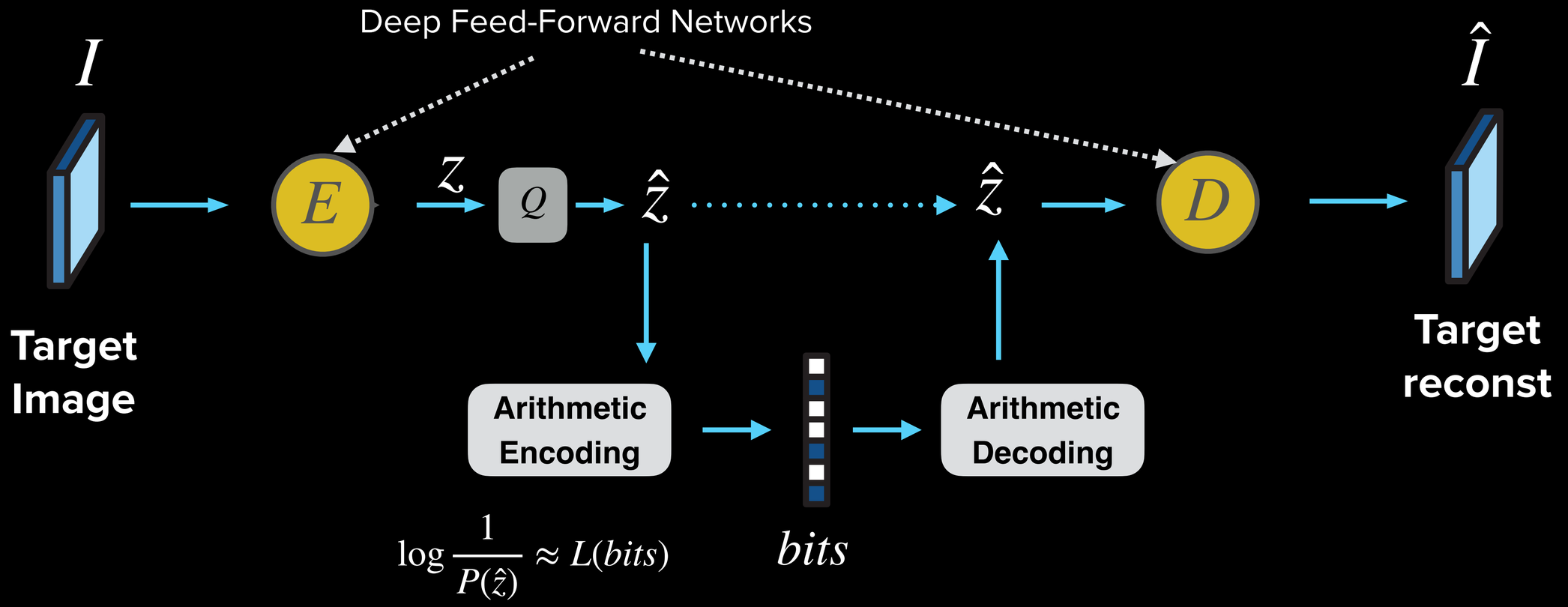


Learned Image Codecs



Question: How do we train the parameters?

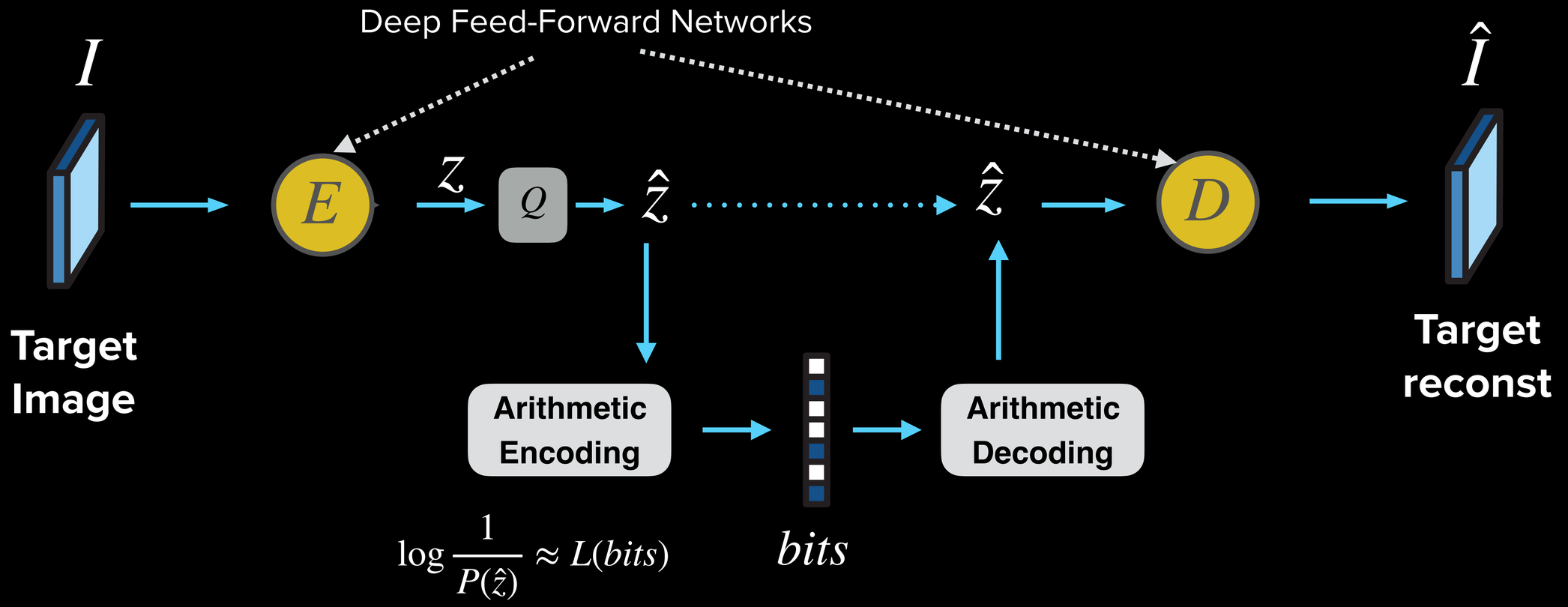
Learned Image Codecs



Loss Function = $L(bits) + \lambda d(I, \hat{I})$

Rate

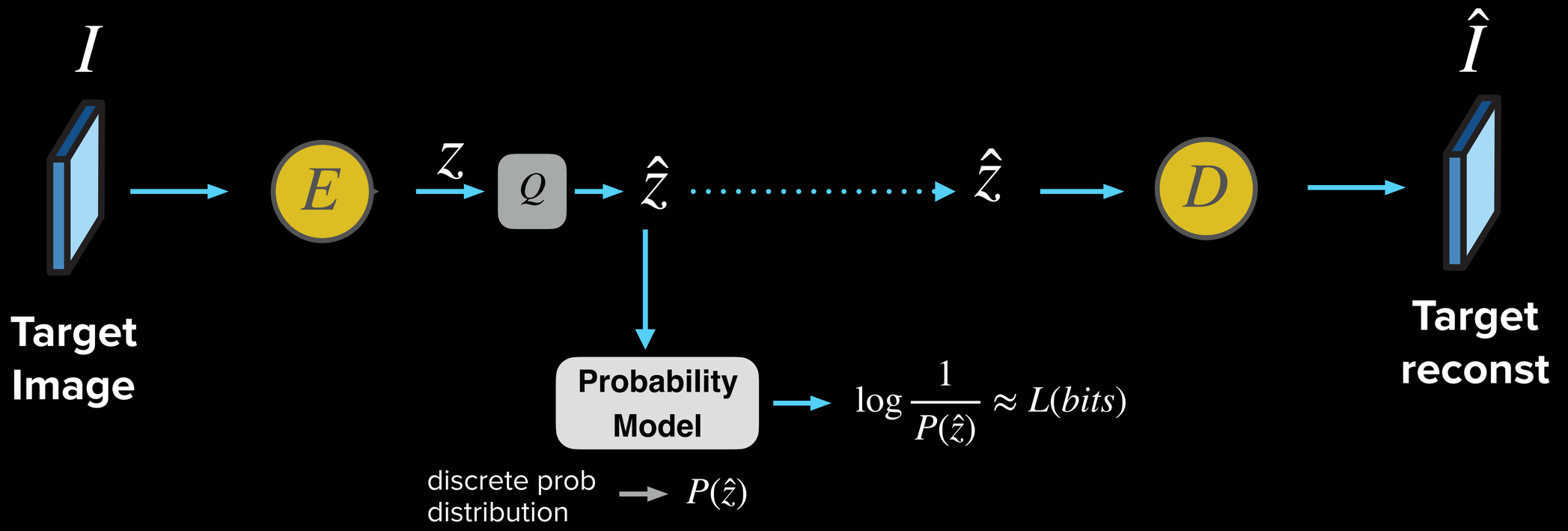
Learned Image Codecs



Loss Function = $L(bits) + \lambda d(I, \hat{I}) \approx \log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

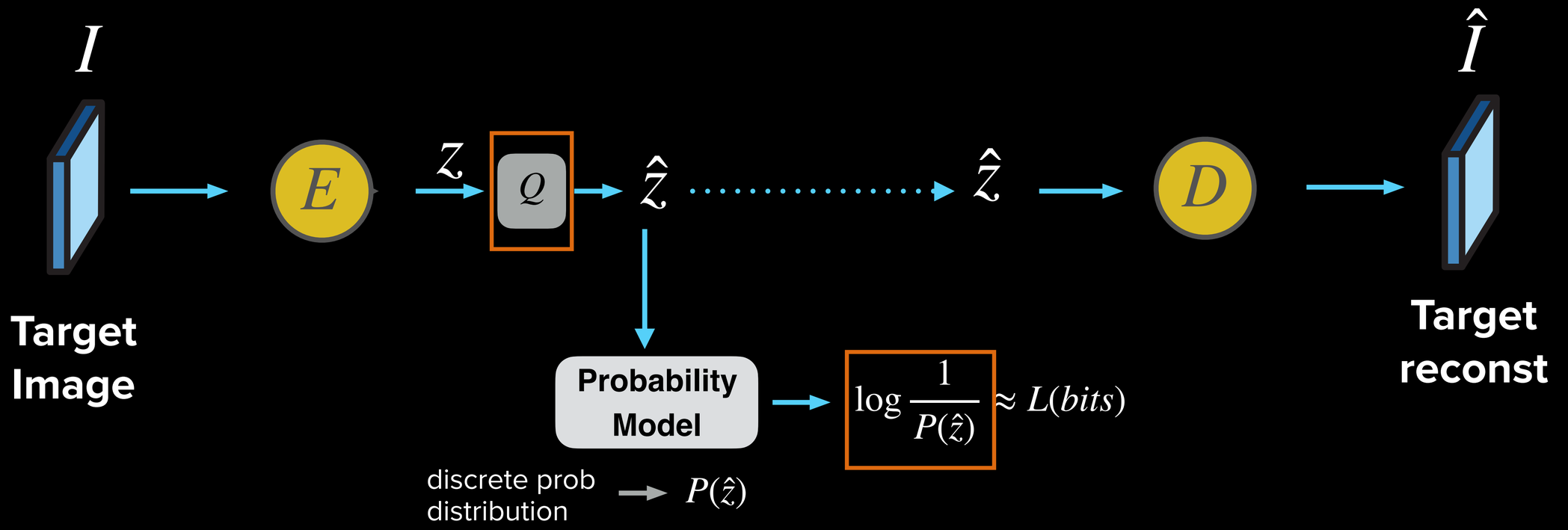
Rate

Learned Image Codecs



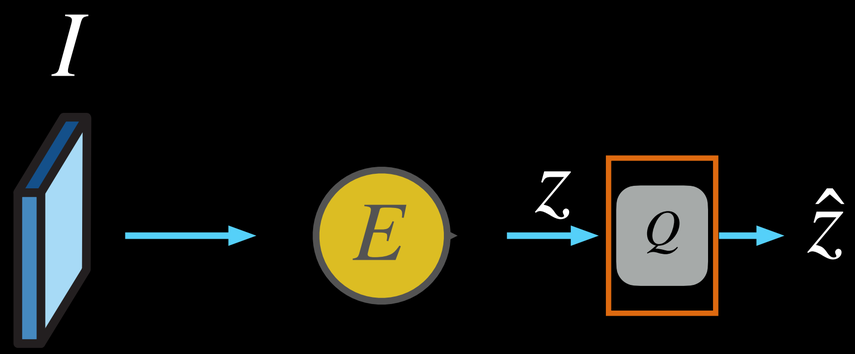
Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

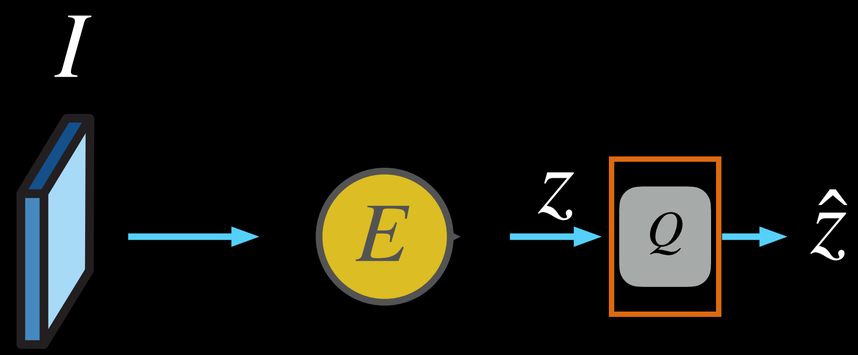


Target Image

- **Quantizer** -> $Q([2.3,3.7]) = [2,4]$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



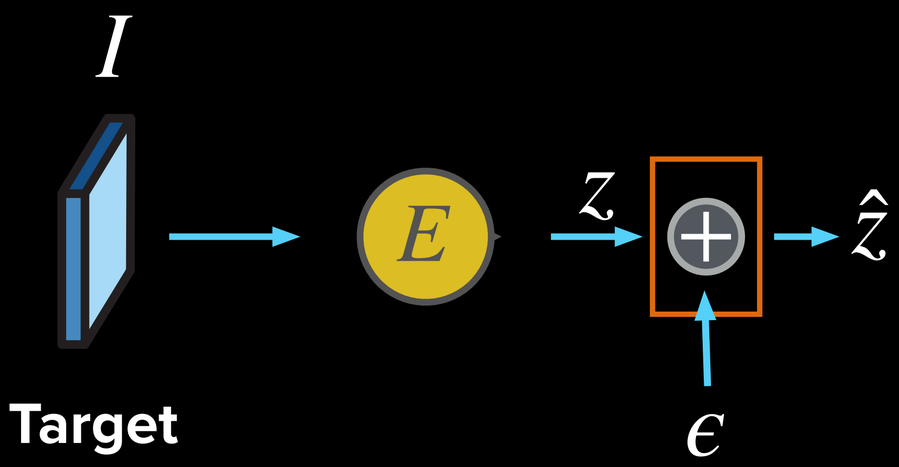
Target Image

- **Quantizer** -> $Q([2.3,3.7]) = [2,4]$
- **Workaround-1:** model the quantizer as adding noise during training

  $\hat{z} = z + \epsilon$, where $\epsilon \sim U(-0.5,0.5)$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

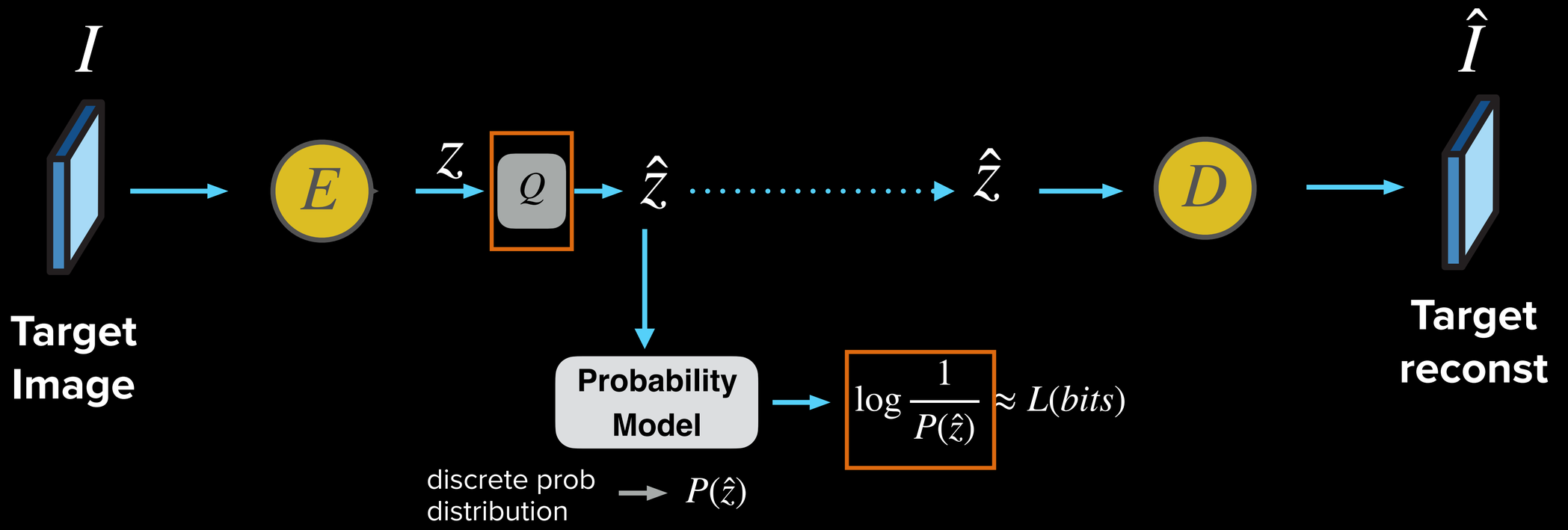


- **Quantizer** -> $Q([2.3,3.7]) = [2,4]$
- **Workaround-1:** model the quantizer as adding noise during training

  $\hat{z} = z + \epsilon$, where $\epsilon \sim U(-0.5,0.5)$

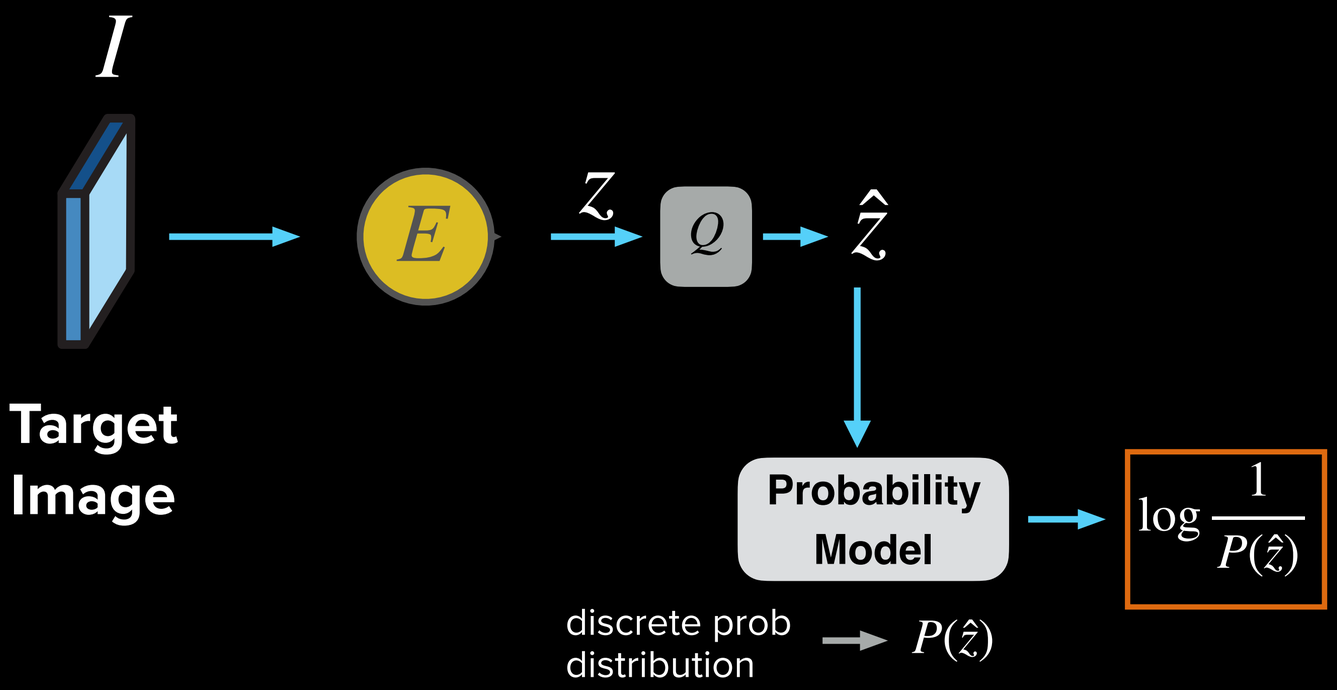
Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

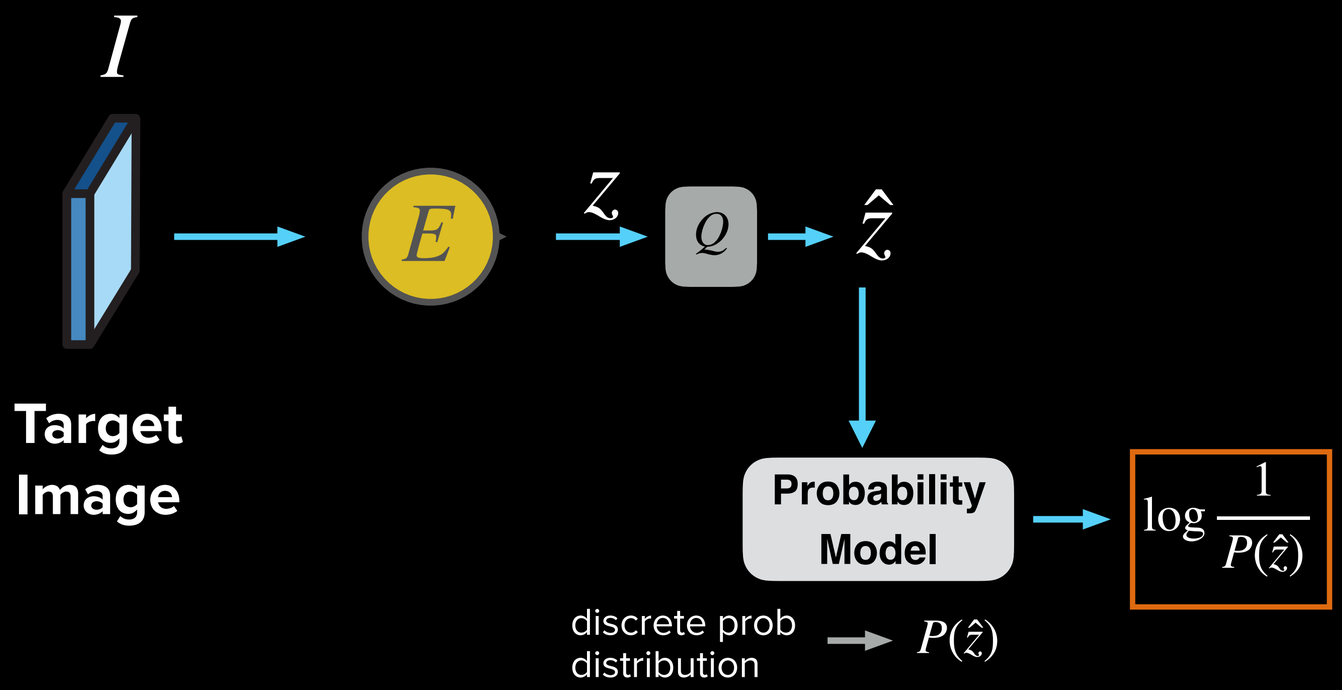
Learned Image Codecs



- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

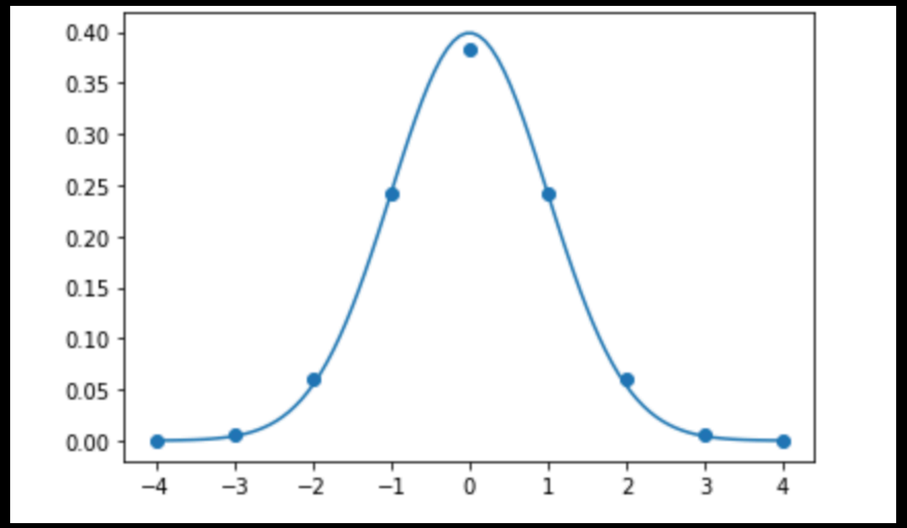


- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

$$P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs

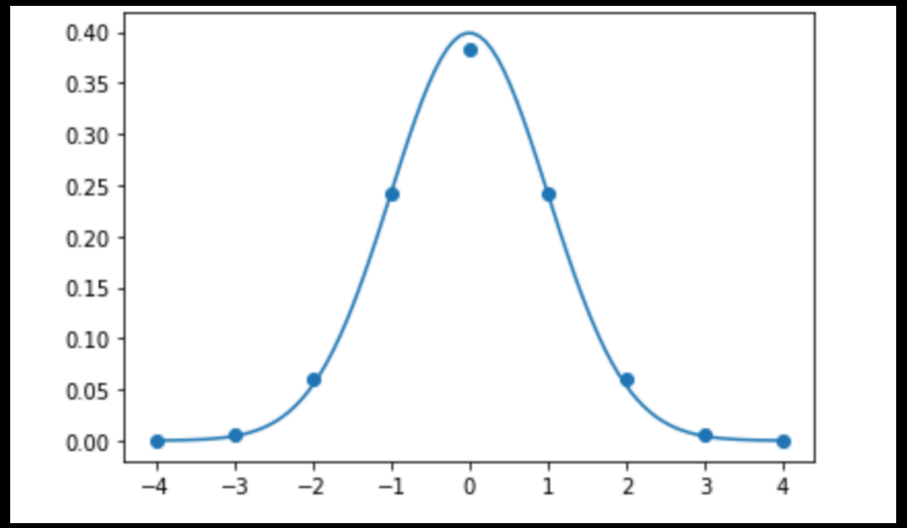


$$P(\hat{z}) = GaussCDF(\hat{z} + 0.5) - GaussCDF(\hat{z} - 0.5)$$

- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$

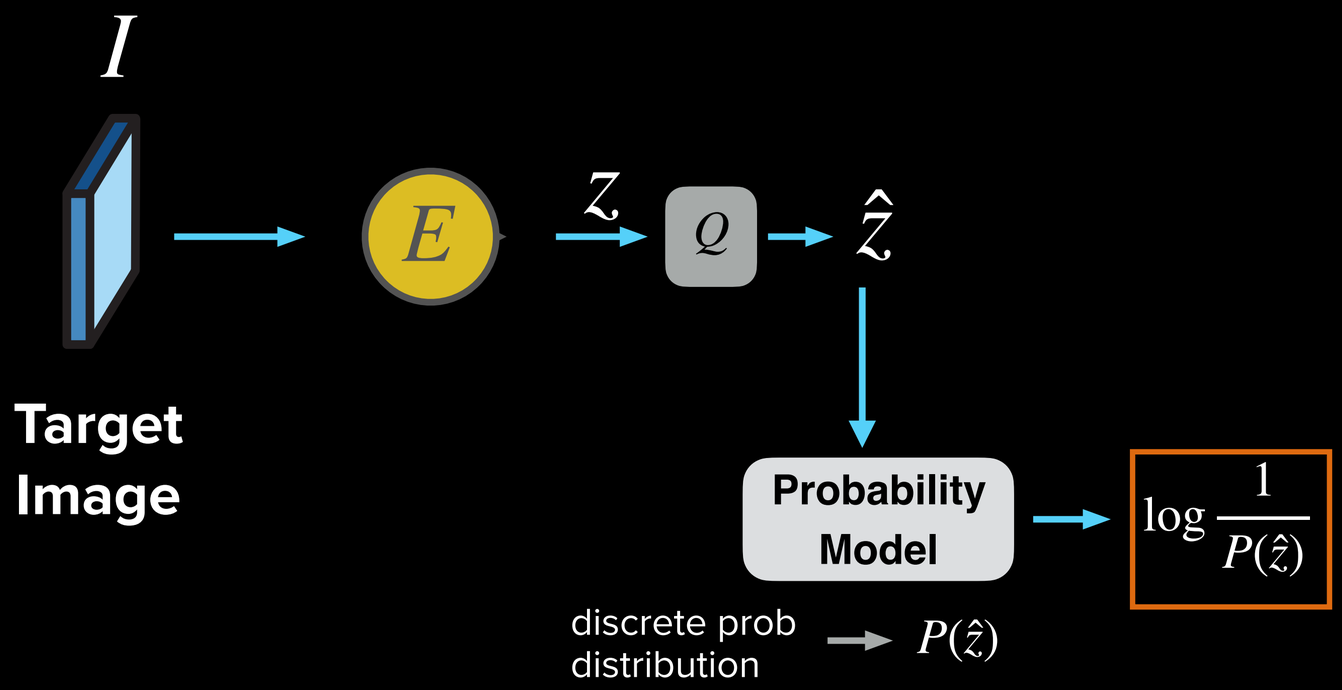
Learned Image Codecs



$$P(\hat{z}) = GaussCDF(\hat{z} + 0.5) - GaussCDF(\hat{z} - 0.5)$$

- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)
  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$
- Gradient is now well defined!
  $\frac{\partial P(\hat{z})}{\partial \hat{z}} = PDF(\hat{z} + 0.5) - PDF(\hat{z} - 0.5)$

Learned Image Codecs



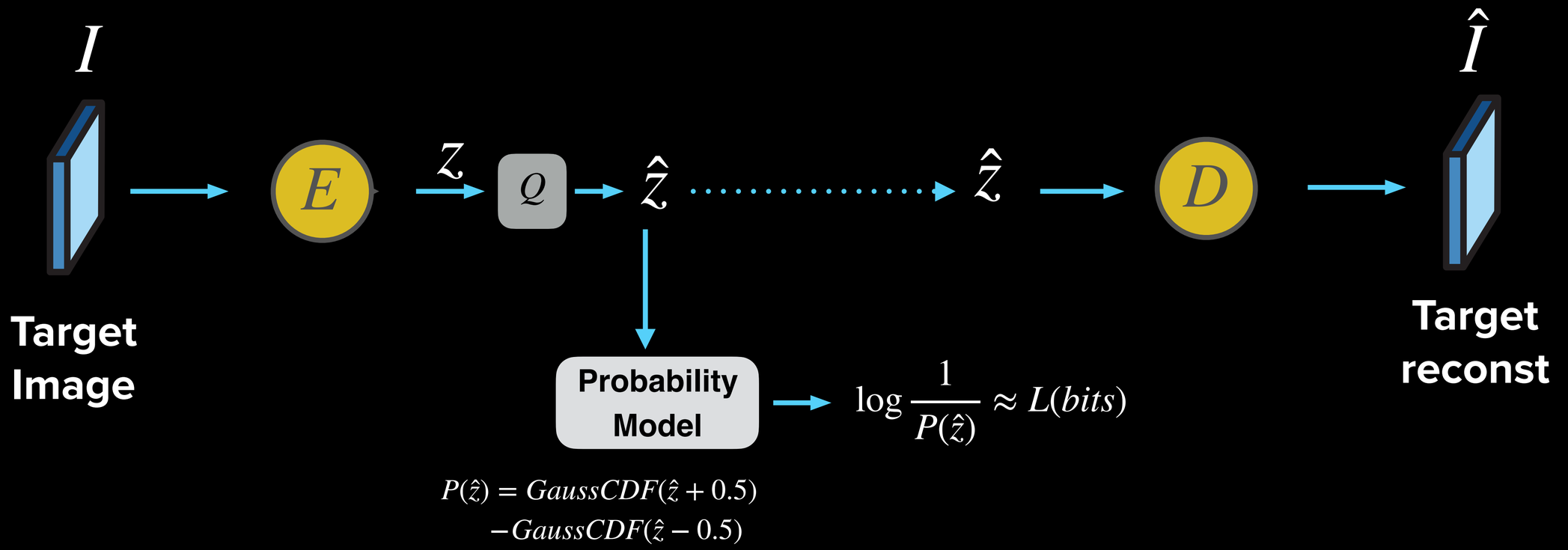
- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$
- Gradient is now well defined!

  $\frac{\partial P(\hat{z})}{\partial \hat{z}} = PDF(\hat{z} + 0.5) - PDF(\hat{z} - 0.5)$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

End-to-End Learned Image Codec



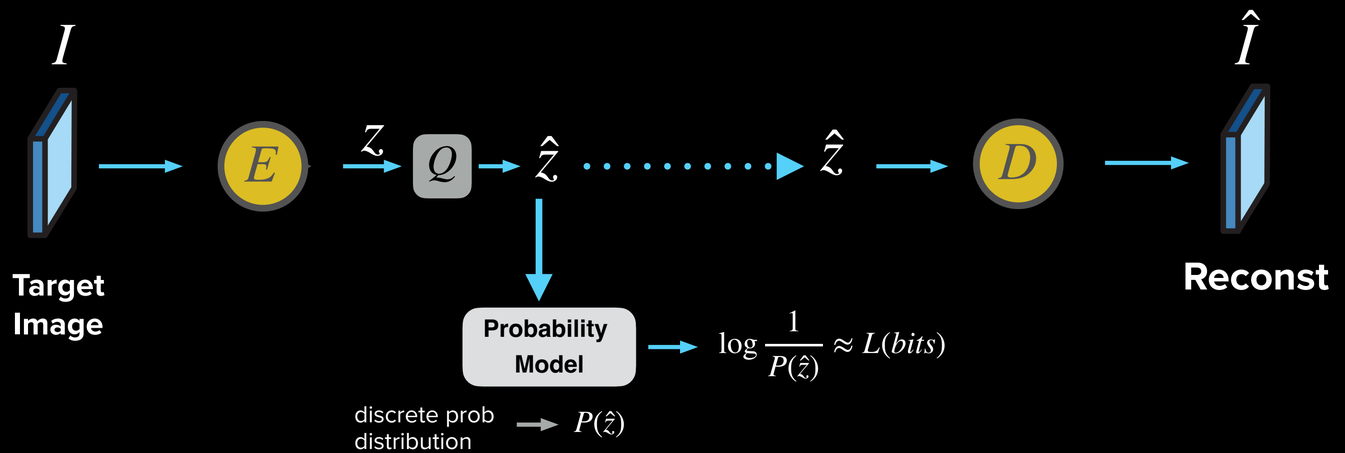
Possible to train end-to-end!

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Example -> MNIST

- https://colab.research.google.com/drive/1s8eL9domVoiUPYsqbIMQz5nKNGbwOESI?usp=sharing

Learned Image Codecs



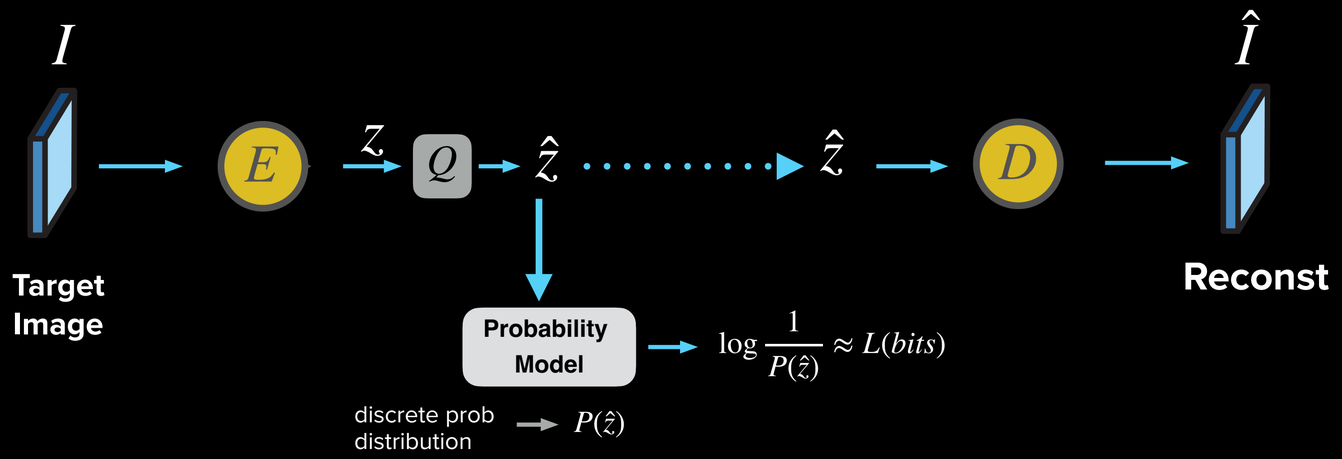
Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

No hand-tuning parameters

Can “learn” the parameters

Possible to train end-to-end!

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

No hand-tuning parameters

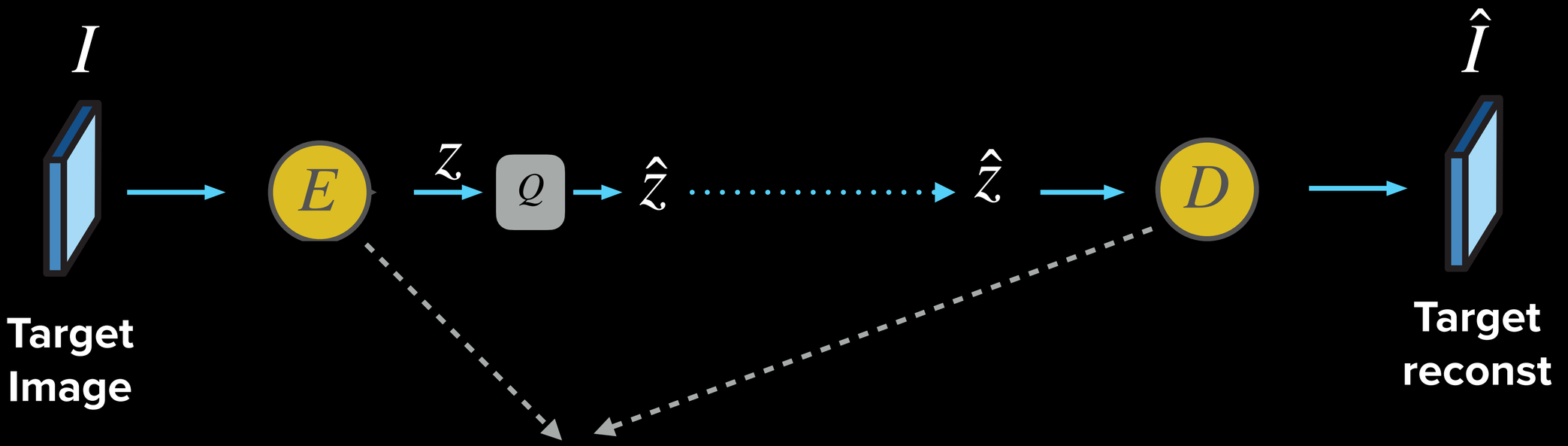
Can “learn” the parameters

Better distortion-Model separation

Easy to tune model to different distortion metrics

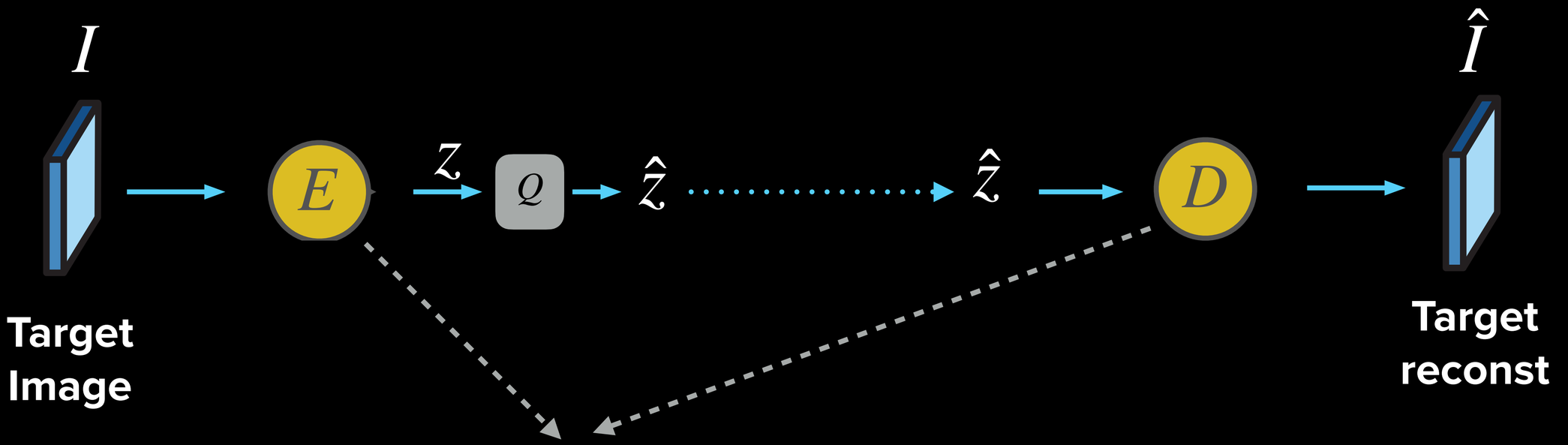
Possible to train end-to-end!

Design Decisions -> Backbones



- **Question:** How can the same model work for any image sizes?

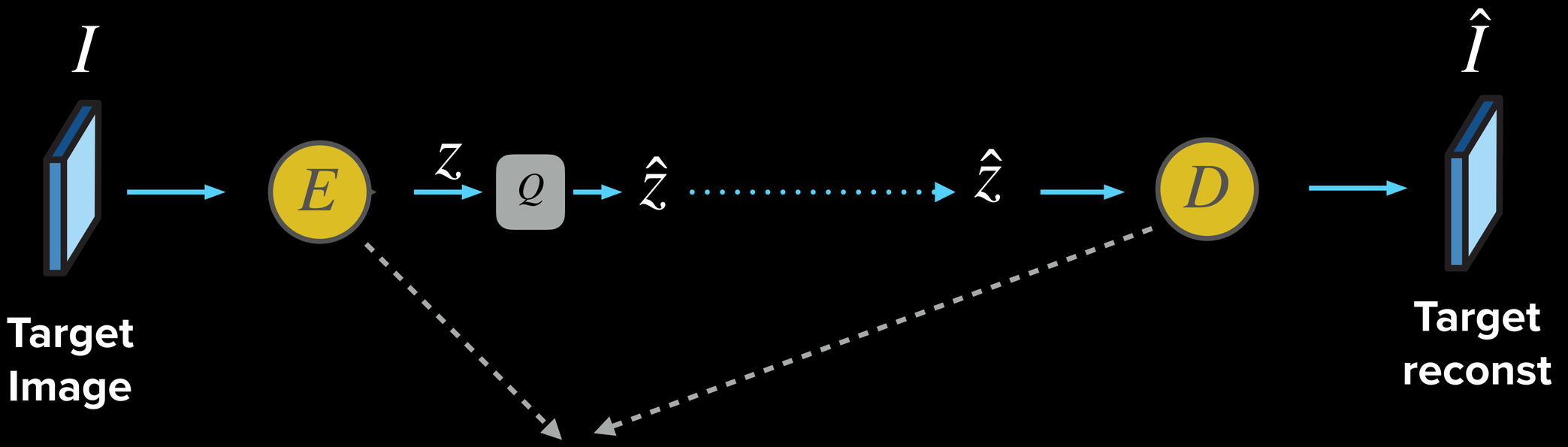
Design Decisions -> Backbones



- **Question:** How can the same model work for any image sizes?

  - Only use Conv, Deconv ... (no Fully Connected Layers)

Design Decisions -> Backbones

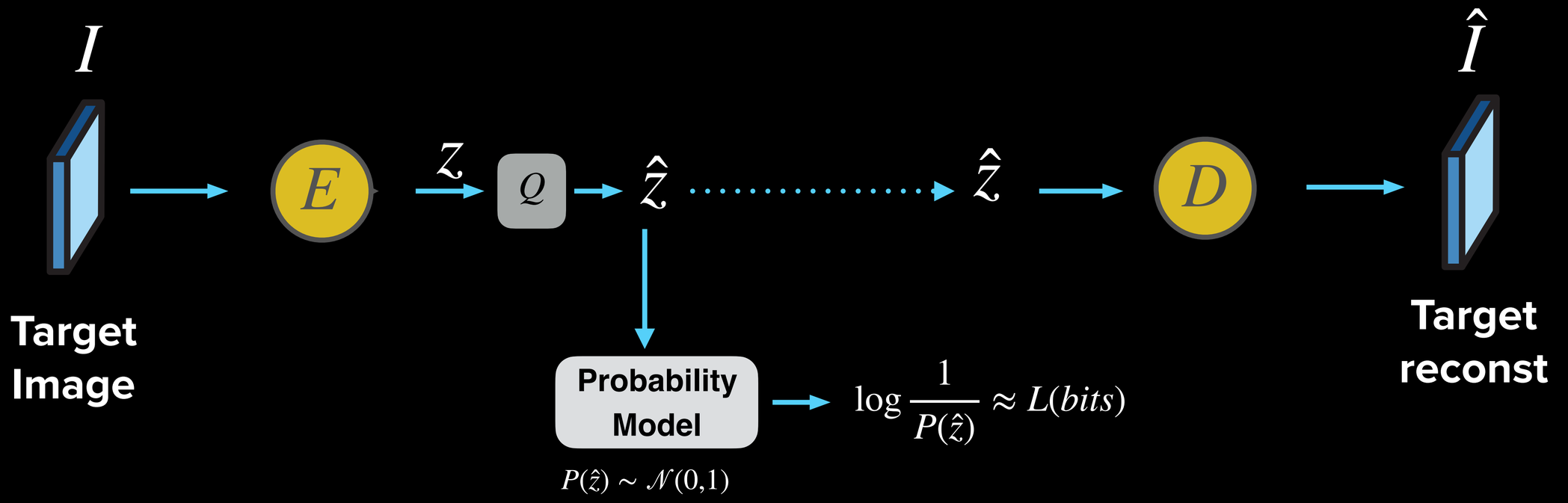


- **Other Improvements:**

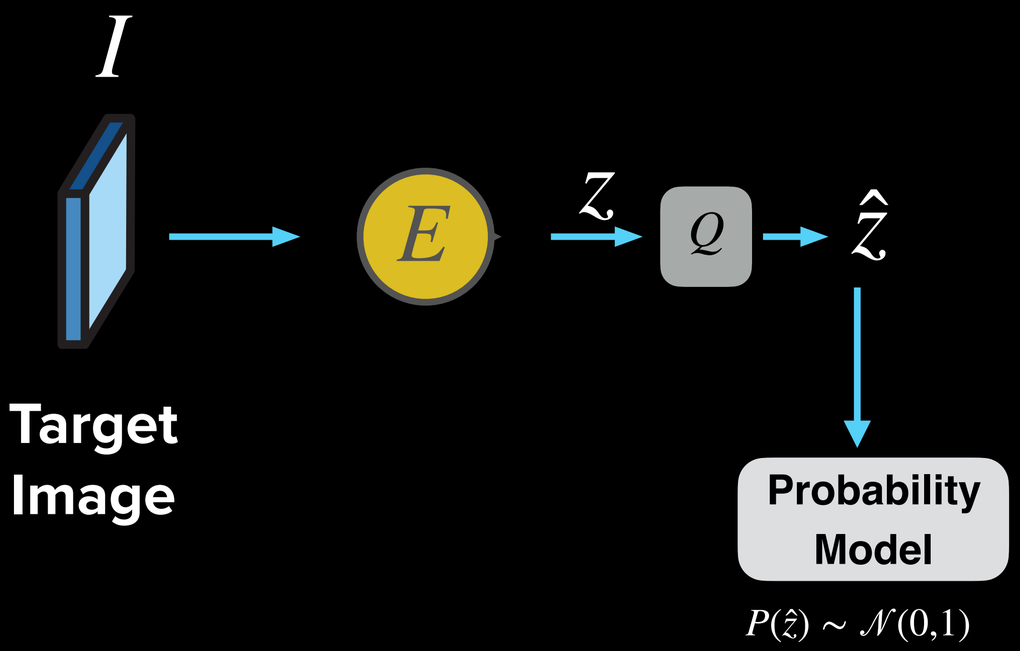
  - Multiscale Encoder/Decoder: [Rippel et. al. 2018]

  - using GDN non-linearity: [Balle, 2017/18]

Design Decisions -> Probability Model



Design Decisions -> Probability Model

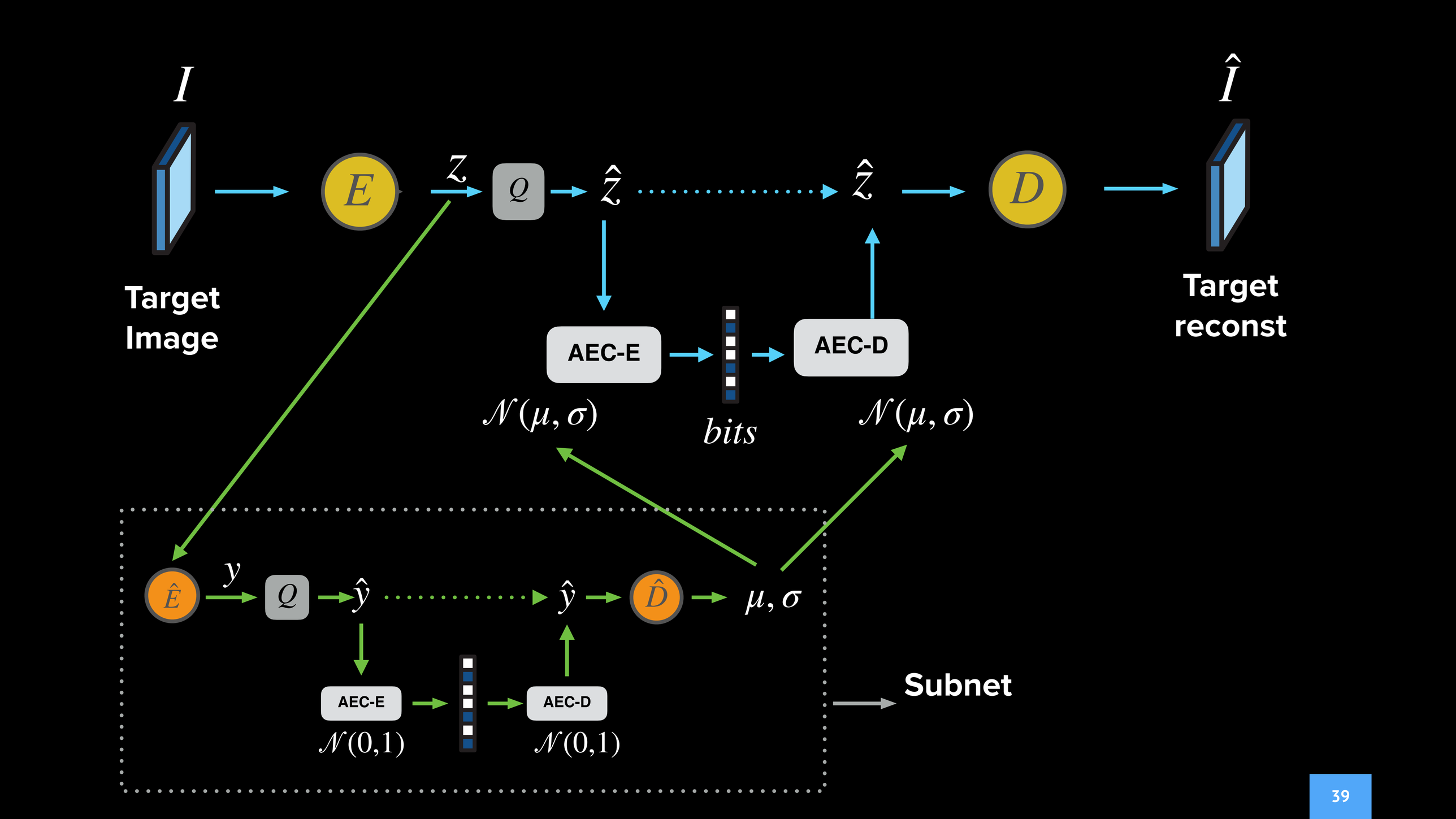


- More complex Probability models
- $PDF(\hat{z}_i) \rightarrow \mathcal{N}(\mu_i, \sigma_i)$,

  i.e: μ, σ are different per element of $\hat{z}$
- Need to now encode μ, σ tensors:

  - *Hyperprior* approach

  [Balle, ICLR18]


I
$\hat{I}$
E
z
Q
$\hat{z}$
$\hat{z}$
D
Target Image
Target reconst
AEC-E
AEC-D
$\mathcal{N}(\mu, \sigma)$
$bits$
$\mathcal{N}(\mu, \sigma)$
$\hat{E}$
y
Q
$\hat{y}$
$\hat{y}$
$\hat{D}$
μ, σ
AEC-E
AEC-D
$\mathcal{N}(0,1)$
$\mathcal{N}(0,1)$
Subnet

Design Decisions -> Probability Model

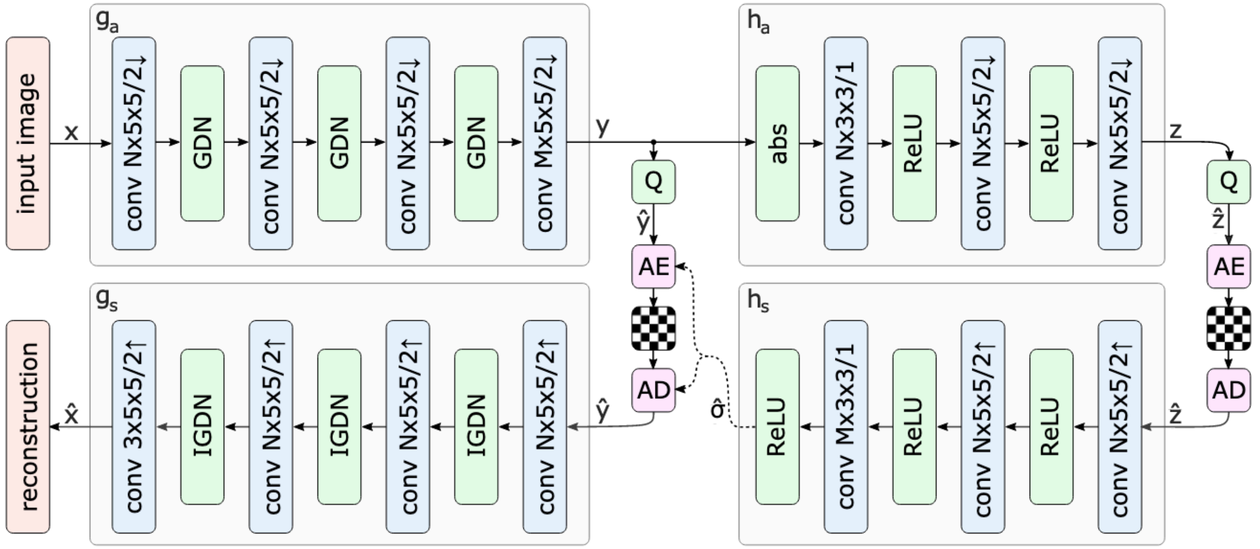


Figure 4: Network architecture of the hyperprior model. The left side shows an image autoencoder architecture, the right side corresponds to the autoencoder implementing the hyperprior. The factorized-prior model uses the identical architecture for the analysis and synthesis transforms g_a and g_s. Q represents quantization, and AE, AD represent arithmetic encoder and arithmetic decoder, respectively. Convolution parameters are denoted as: number of filters $\times$ kernel support height $\times$ kernel support width / down- or upsampling stride, where $\uparrow$ indicates upsampling and $\downarrow$ downsampling. N and M were chosen dependent on λ, with $N = 128$ and $M = 192$ for the 5 lower values, and $N = 192$ and $M = 320$ for the 3 higher values.

Design Decisions -> Probability Model

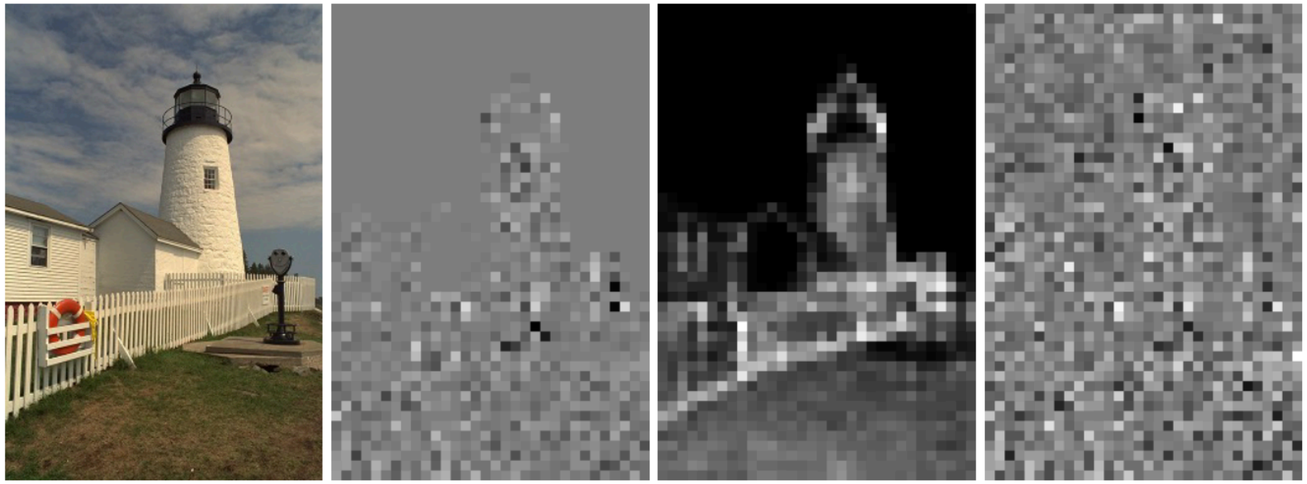
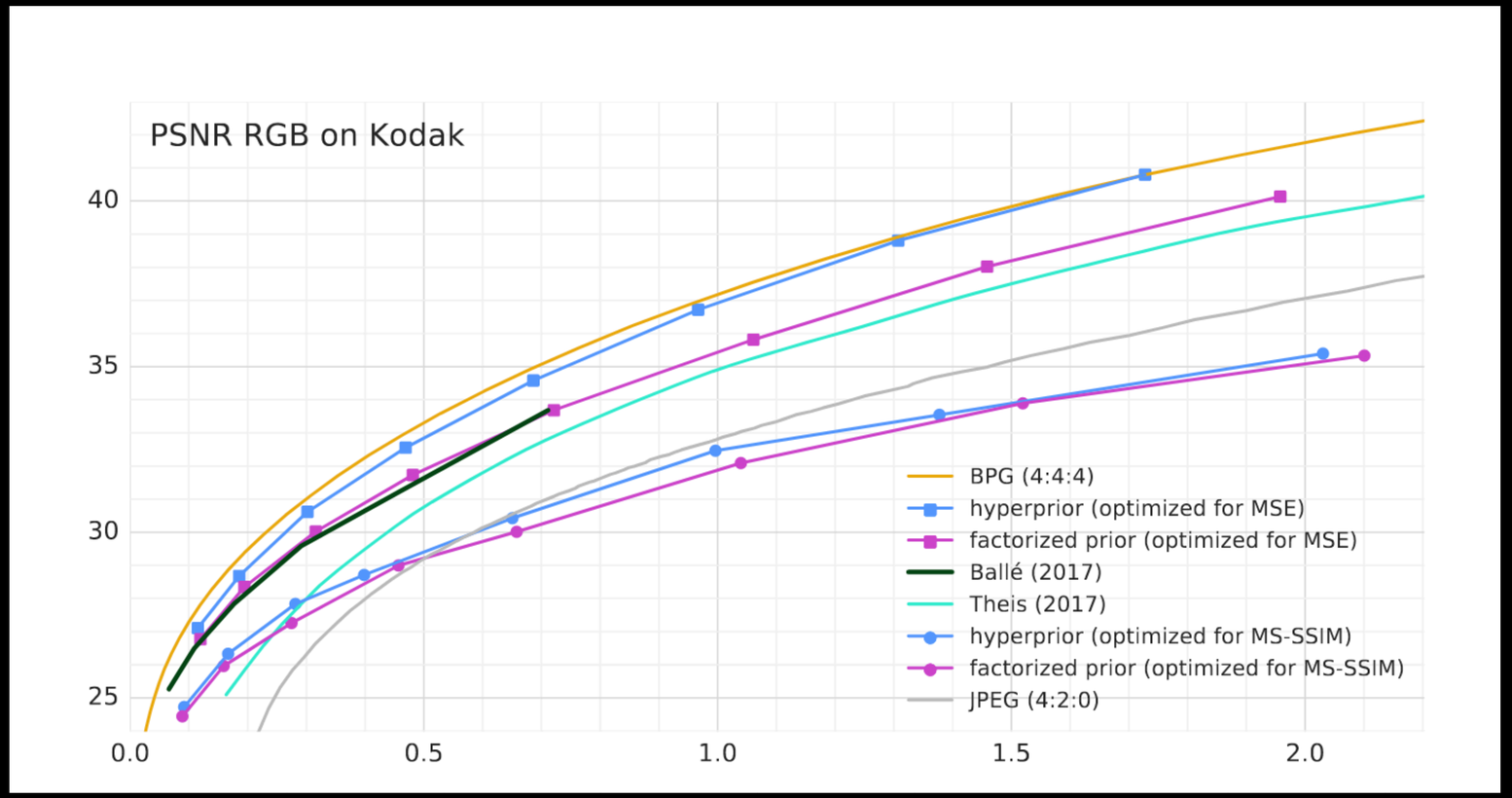
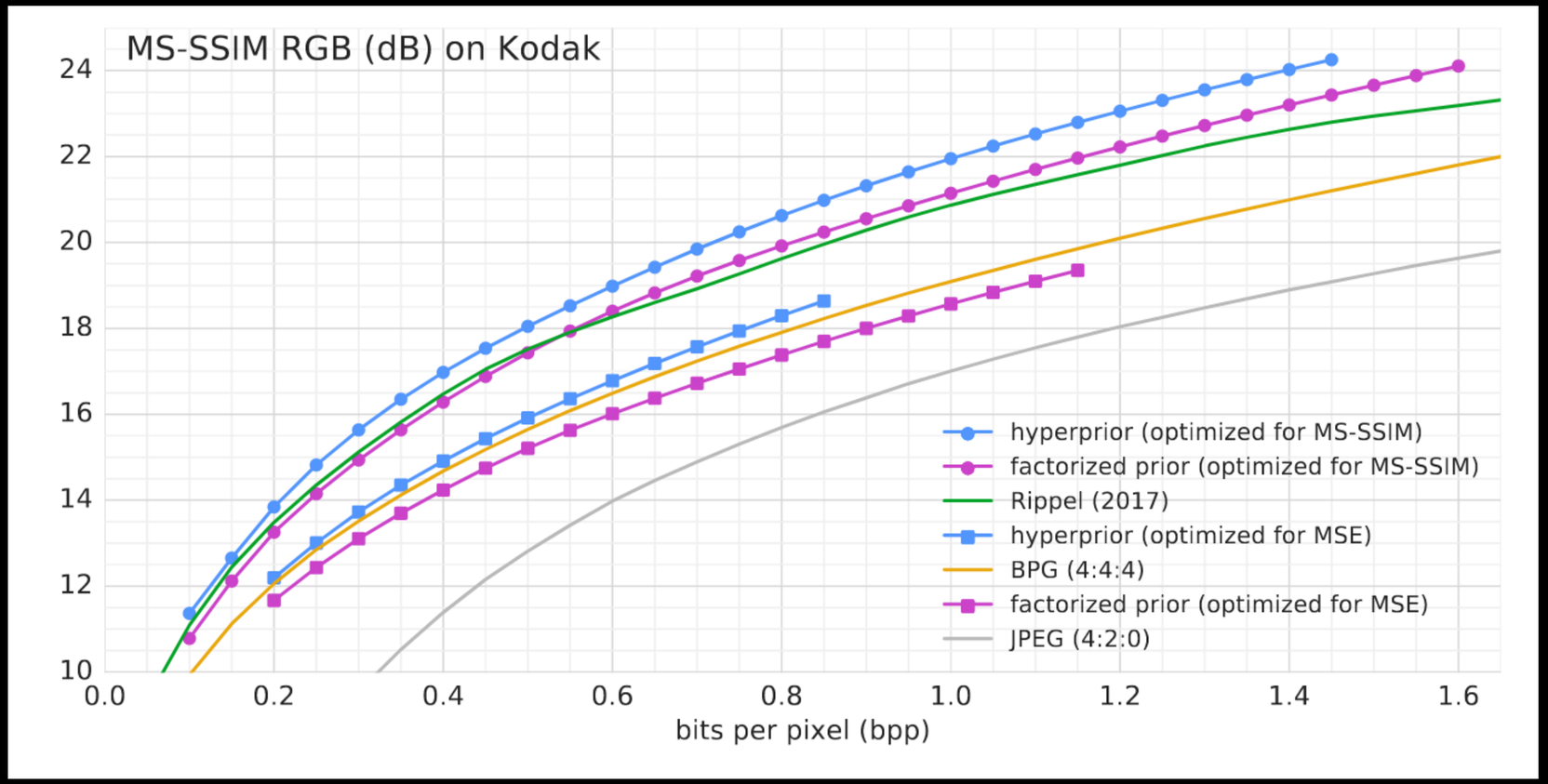


Figure 2: Left: an image from the Kodak dataset. Middle left: visualization of a subset of the latent representation $\boldsymbol{y}$ of that image, learned by our factorized-prior model. Note that there is clearly visible structure around edges and textured regions, indicating that a dependency structure exists in the marginal which is not represented in the factorized prior. Middle right: standard deviations $\hat{\boldsymbol{\sigma}}$ of the latents as predicted by the model augmented with a hyperprior. Right: latents $\boldsymbol{y}$ divided elementwise by their standard deviation. Note how this reduces the apparent structure, indicating that the structure is captured by the new prior.

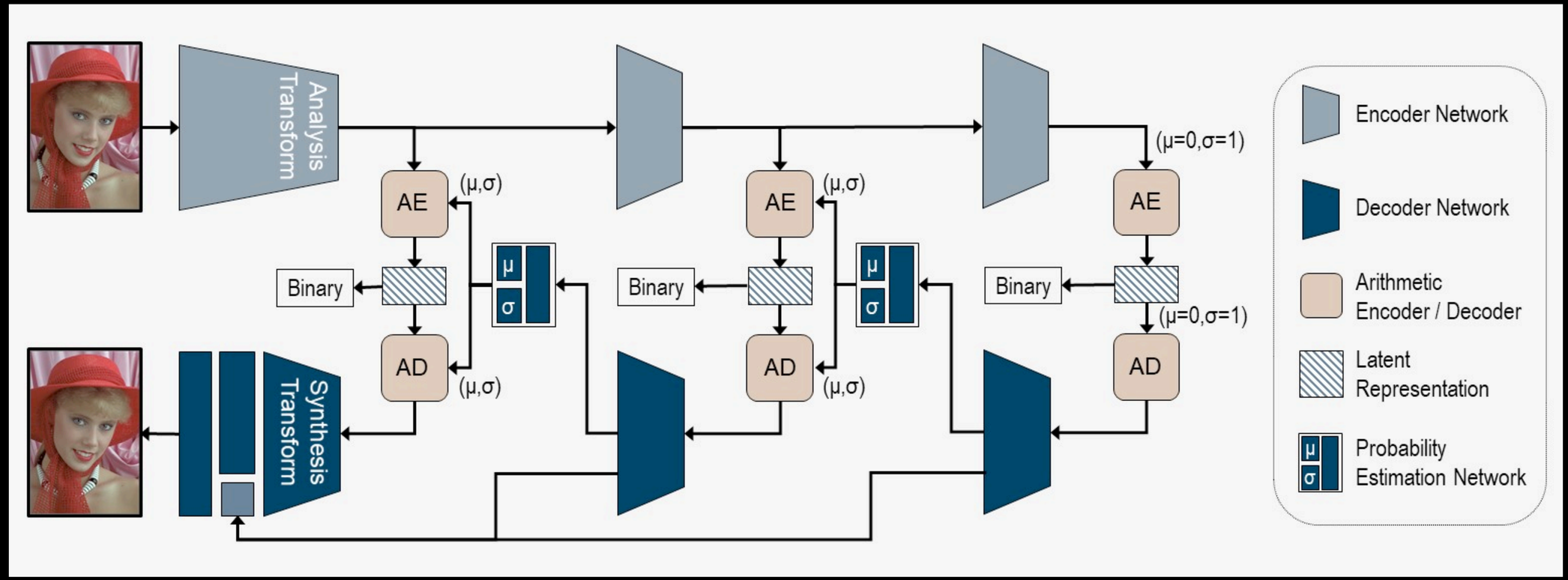
Compression Performance



Compression Performance



Hyper-hyper-prior models



- https://huzi96.github.io/coarse-to-fine-compression.html

Hyper-hyper-prior models

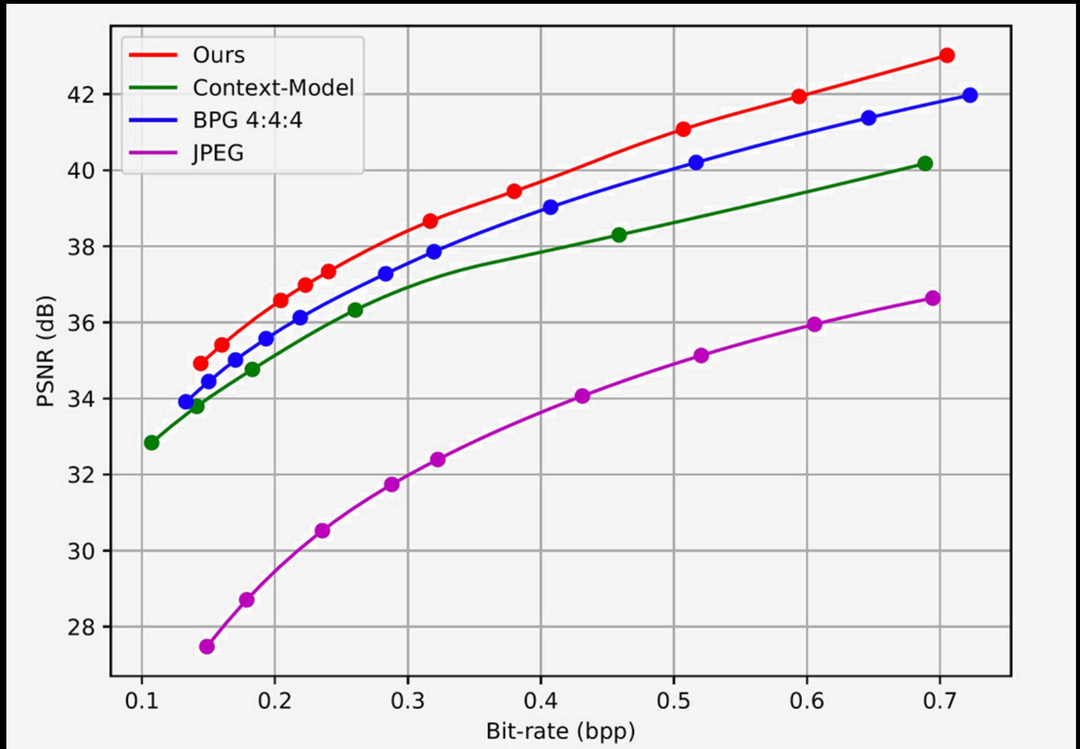
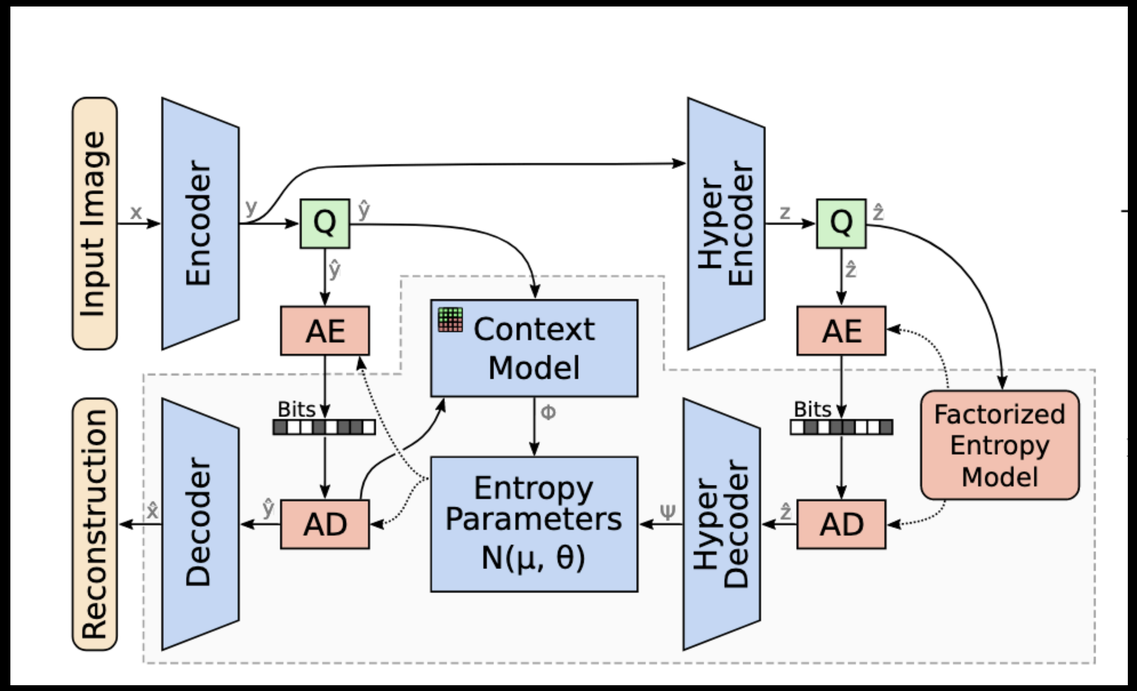


Fig. 4. RD Curve on Tecnick Dataset.

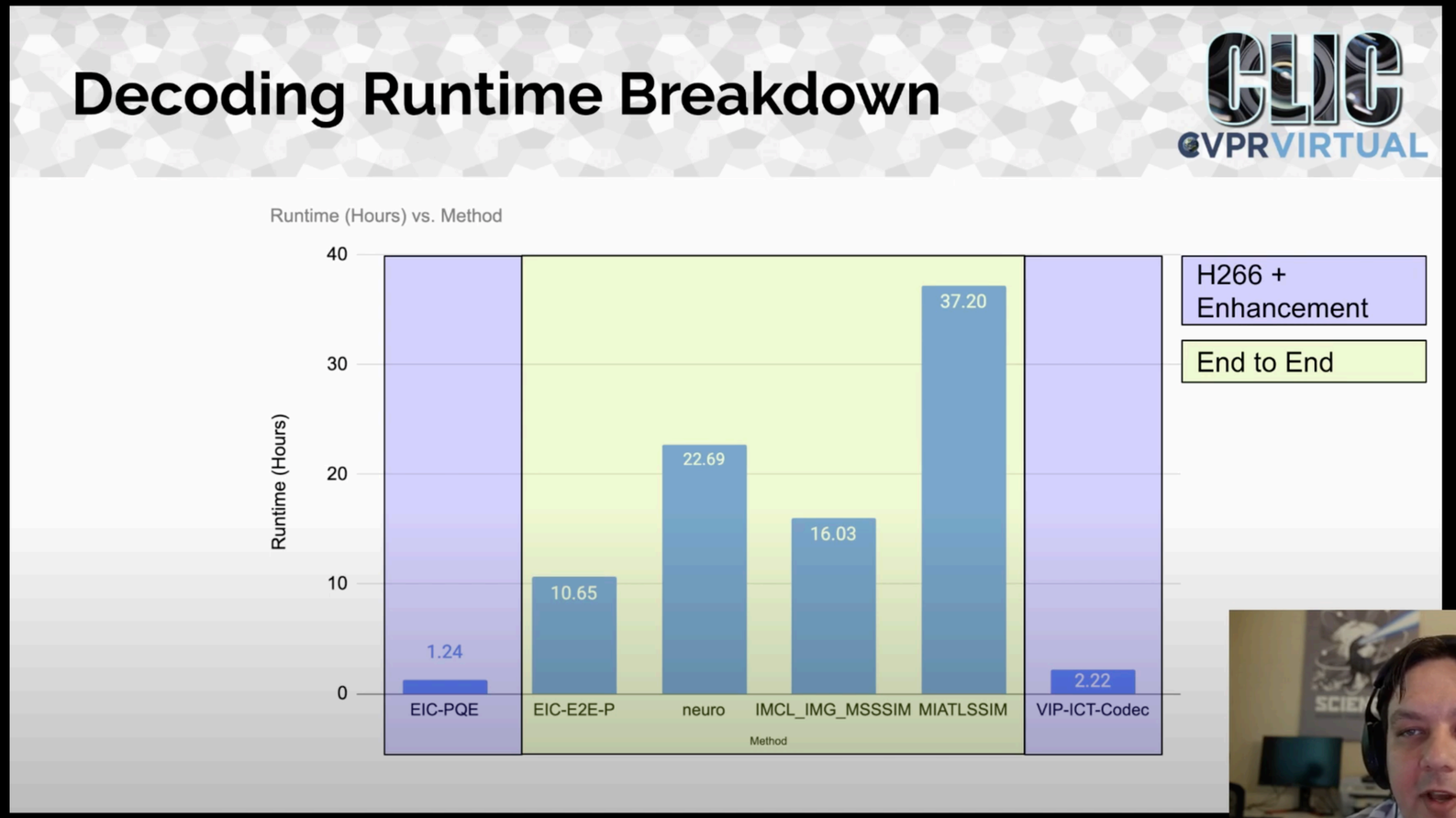
- https://huzi96.github.io/coarse-to-fine-compression.html

Hyperprior with Context model



Joint Autoregressive and Hierarchical Priors for Learned Image Compression [2019, Minnen et.al]

Speed!



Slide from Dr. Toderici's talk at CVPR20

Speed - Design Decisions

- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

Speed - Design Decisions

- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

- **For example:** [ELF-VC, Rippel et.al. 2021, ArXiv]

  real-time HD 720 decode on mid-range GPU

  - VGA 640x480: encode @ 47 FPS, decode @ 91 FPS

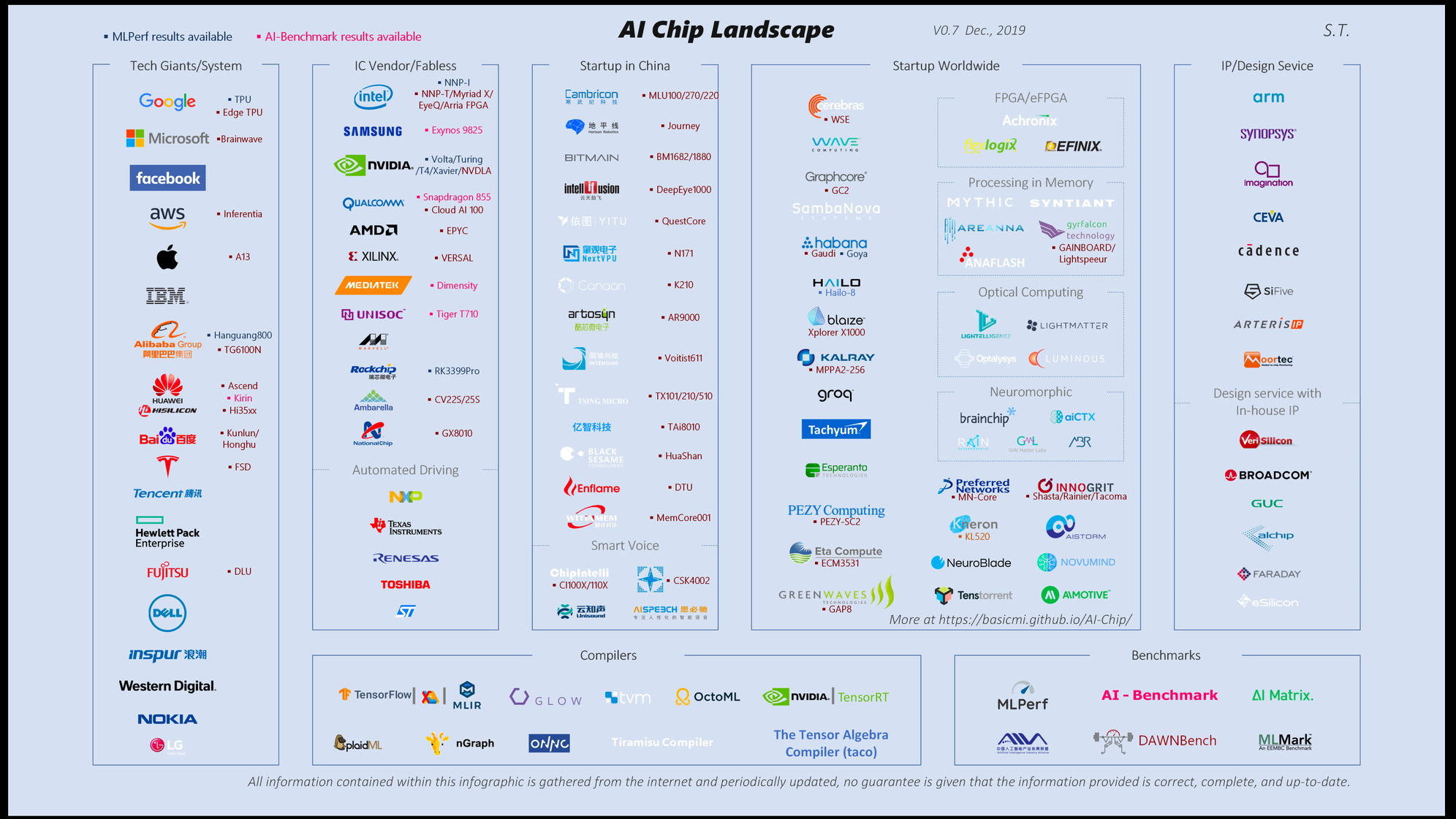
  - HD 1280x720: encode @ 19 FPS, decode @ 35 FPS

Speed!

- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

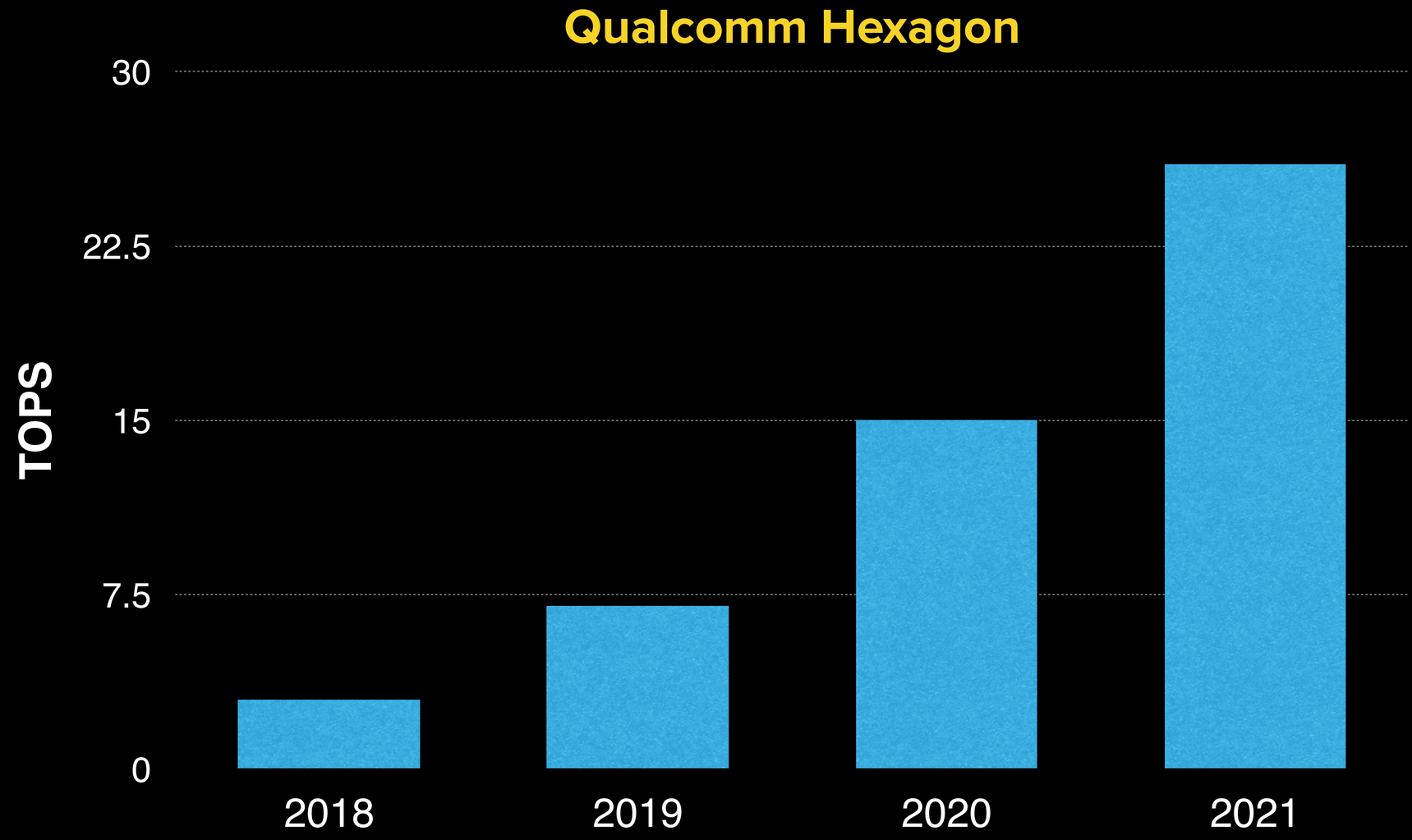
- **Faster Hardware:** Hardware support for NN keeps improving year by year

Hardware getting better!



All information contained within this infographic is gathered from the internet and periodically updated, no guarantee is given that the information provided is correct, complete, and up-to-date.

Hardware getting better!

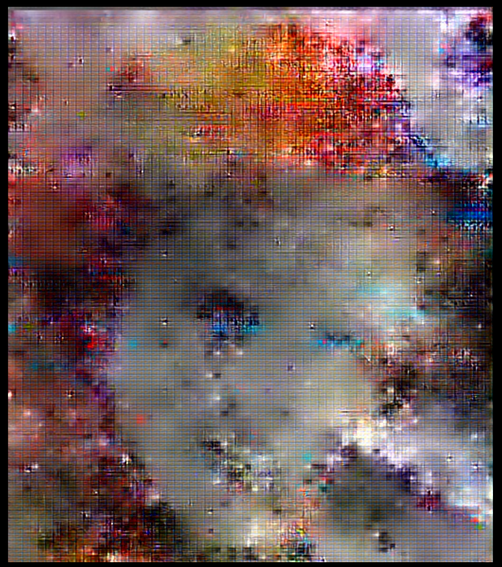


Determinism

- Different hardware ->

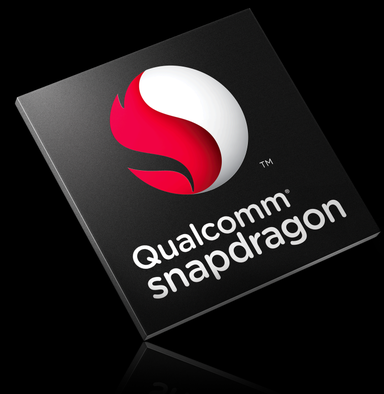
  different floating point implementation

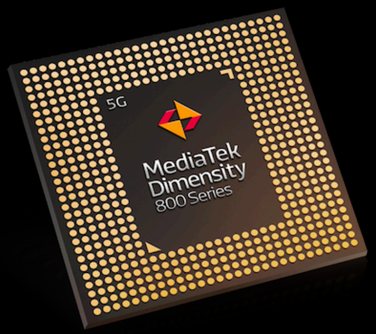
  -> catastrophic failures!



Determinism

- Encoding/decoding must yield exactly the same outputs, irrespective of hardware architecture





- Floating point (FP32/16) models don’t work:

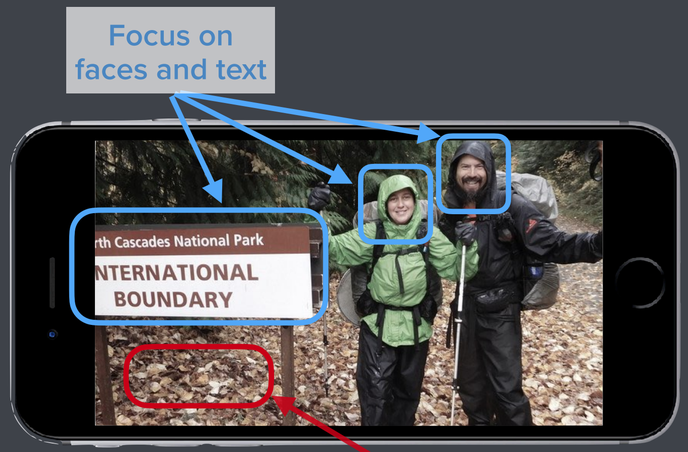
  Model Quantization necessary [E.g: Ballé 2019]

ML Offers Adaptivity Not Possible With Traditional Codecs

Domain-aware

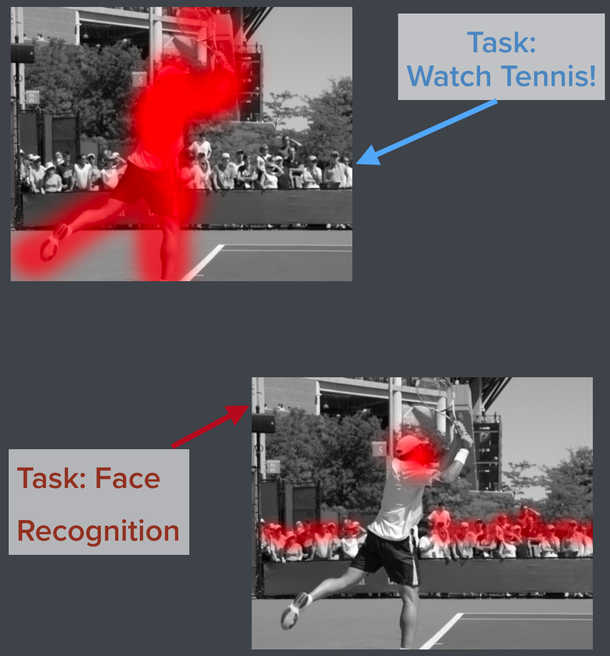
Custom codec for each domain

Content-aware



Focus on areas of high importance

Task-aware



Computation on compressed representation

Summary

- ML-based codecs are flexible and adaptable
- ML-based methods will likely form the basis of future compression, representation and analytics of video
- We are at the beginning of this trend!

What Next? -> Human Perception

- ML-based image compression can be trained with any given loss metric

  (MSE, L1 …)

- How can we go about designing a perceptual loss metric?